AN APP TO USE TFGM OPEN DATA

A DISSERTATION SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF MASTER OF SCIENCE
IN THE FACULTY OF SCIENCE AND ENGINEERING

2018

By
Jung Huang
(10159031)

Supervisor: Konstantin Korovin
School of Computer Science

Contents

Word Count: 15318

List of Tables

List of Figures

Abstract

Nowadays, open data has become a tremendous resource that allows the public to collect a wide range of different types of data to provide a service or build an application. TfGM (Transport for Greater Manchester) is one of the government open data services derived from data.gov.uk, which provides an API (Open Data Service Version 2.0) about the transport network in Greater Manchester. In order to extend the value of transportation data, this dissertation focuses on providing trip plan recommendations for travelers. An Android mobile application is built to perform this task through integrating multiple information from different resources. Compare to other trip planners, the main feature of this application is to simplify the process of trip planning that users can obtain a complete itinerary by entering some conditions rather than organizing the attractions, time, and route by themselves. To achieve this goal, several pieces of research and objectives are described in the following chapters. All of the data sources of this project are from web service APIs in JSON format so that they need to be parsed into human-readable text. Besides, the trip plan route is optimized by using Genetic Algorithm to solve Traveling Salesman Problem. Finally, SimpleNLG, a Natural Language Generation library, is used to produce a clear trip plan summary. The whole project is developed based on agile development methodology, Model-View-Controller pattern, and Test-driven development process. It starts with defining functional and non-functional requirements, and considerations, then implementing in iterations with small releases. Testing results show that the implementation of the application is appropriate and evaluation results show that the functionalities of the application are acceptable; however, it can be improved by adding more features or refining the existing functionality.

Keywords: *Trip plan recommendations; Android mobile apps; Test-driven development; Agile software development; Model-View-Controller pattern; Web services; Natural Language Generation; Genetic Algorithm for Traveling Salesman Problem*

Declaration

No portion of the work referred to in this dissertation has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Acknowledgements

First of all, I would like to thank to my supervisor, Dr. Konstantin Korovin for providing continuous guidance, general support and helpful advice throughout the development of this project. I would also like to thank my family and friends for their unwavering support and continuous encouragement, the best fuel I could ever hope for throughout this journey. I also want to express gratitude to the School of Computer Science for providing rich academic resources to support the learning process. Finally, I would also like to thank everyone who contributed directly or indirectly towards the successful completion of this dissertation. Without these helps, I would not have completed the report.

Chapter 1

Introduction

This chapter introduces the motivation of this project and what research questions are tried to be answered through this dissertation. Moreover, the aim and objectives of this project are explained. Project deliverable is briefly described and the dissertation structure is presented at the end of this chapter.

1.1 Motivation

Open Definition defines ”open data” as that ”Open data is data that can be free to use, reuse, and redistribute it subject only, at most, to the requirement to attribute and share-alike.” [1] The aim of open data is to make data more transparent and foster innovation, it will be easier for the public to make decisions and suggestions about knowledge or government policies based on detailed information. There are many kinds of open data, such as science, finance, weather, etc., one of the most important forms is open government data, which can be used to learn more about how government works or carry out research. For example, data.gov.uk [2] is a UK Government project launched in 2009, contains over 40,000 datasets from various central government departments and raw data for people to develop useful applications, including environment, government spending, education, health, transport, etc.

Transport for Greater Manchester (TfGM) is one of the services in data.gov.uk which provides an API (Open Data Service Version 2.0) about the transport network in Greater Manchester. The goal of TfGM is to encourage the development of useful applications that provide more information for travelers to make smarter choices. TfGM is in JSON format, which is unfamiliar to the users, in order to help them understand this information, one of the main challenges of this project is to transfer JSON format into a human-readable format.

Since transport is one of the most important factors for a trip plan, in order to provide the more practical features for travelers, this project not only provides human-readable traffic information but also integrates with other data from different resources to generate a trip plan for users. Nowadays, the availability of trip planning app has grown significantly, such as TripAdvisor, Tripit, travelers prefer to plan their itinerary on the mobile device before they go. Most of the current trip planners are focused on attraction recommendation, users still need to organize the itinerary manually, such as deciding where to go and making the sequence of destinations, which is time and effort consuming. The purpose of this project is to simplify the process of trip planning, filling some preferences is the only step to generate a trip plan.

1.2 Research Questions

In an attempt to focus the study, determine the research methodology and achieve the goal of the project, the following questions, which are used as the research questions:

- What is a good way to integrate TfGM open data with other open data, such as google map open data?
- What kinds of features are necessary and appropriate for a trip plan?
- What is a good way to present auto-generated trip detail as human-written?
- What is a suitable way to implement the route optimization in this project?

1.3 Aim and Objectives

The aim of this project is to build a mobile phone app that queries the results from multiple APIs depends on user's preference and integrates these results to generate a trip plan. This plan includes plan summary, transport information and attraction details, which is in JSON format; therefore, it should be transferred into human-readable format and presented in a suitable way that travelers find acceptable. Furthermore, the route of the trip plan would be optimized by the specific algorithm, which depends on traveling time. This app provides the suggestions for users who plan their trips or have trouble making decisions.

To achieve the aim of the project, several objectives are defined as project milestones. The objectives are as follows:

- Researching and selecting several APIs, such as TfGM API, google map API that would provide useful information for a trip plan.
- Studying and understanding the OData RESTful API format and how to retrieve required data.
- Comparing and choosing distinctive mobile app platform and studying the app architecture with appropriate User Experience (UX) design.
- Researching and setting some constraints to make trip plan reasonable.
- Studying how to interact data with the map and researching a suitable algorithm for route optimization.
- Studying Natural Language Generation (NLG) to understand the way to transfer semi-structured data into human-written style summary.

1.4 Project Deliverable

- Building an android mobile app to show the useful information and give the precise suggestion for travelers.
- Extract data from APIs and filter with user's preference and constraints to list spots, transport, time and other details.
- Use a specific algorithm to find the optimized path and show the route on a map by google map direction API.
- Use NLG tool, SimpleNLG to transfer data into natural language.
- Generate test suite for the mobile app by different preferences and conditions to see the performance of the app.
- Analyse and evaluate the results by user experience and report recommendations for future work.

1.5 Dissertation Structure

The aim of this dissertation is to provide comprehensive details on the development of Trip Plan mobile application, in order to achieve the objectives and the aim, it is broken down into six major chapters. This chapter has introduced the project in terms of its motivations, research questions, objectives, and deliverable. The remainder of the report, including the background, design, implementation, evaluation, and the conclusion, is structured as follows:

Chapter 2: Background- This chapter provides a complete literature overview of several background topics related to the project. It starts with a description of the web services and web mashups, including the RESTful APIs used in this project, OData, and data-exchange formats. Then, a comparison for the types of mobile apps and Android platform are introduced. Following that, a synopsis of NLG is presented, including tasks, tools, and evaluation methods. Finally, the algorithms for route optimization and several related works are discussed.

Chapter 3: Methodology and Application Design- This chapter begins with the introduction of the agile development methodologies and discusses what is the best process to develop this application. Then the MVC design pattern, gathered requirements, and considerations are described. Finally, the structure of its components and the overview of the implementation process are illustrated.

Chapter 4: Application Implementation- This chapter details the implementation environment and process of developing the system. The overall activity flow and component interactions are explained. Moreover, each Sprint is presented, containing sprint backlog, user interface snapshots, pseudo code for algorithms, burn-up chart, etc. for better understanding and visualization.

Chapter 5: Testing and Evaluation- This chapter presents the testing method and results that TDD is the testing process used in this project. Subsequently, the evaluation method and results are also described that a survey is used to evaluate the application, including system usability, user acceptance, NLG quality, and possible improvement.

Chapter 6: Conclusions and Future Work- This chapter offers the brief conclusions and limitations of the project, and provides insights on any possible future works that could be carried out to enhance the system.

Chapter 2

Background

This chapter begins by introducing the web services which are the main data sources for this trip planning app. In this context, REST, OData, data-exchange formats and several APIs are described. Section 2.2 introduces the web mashups which is the technique used in this project to provide more comprehensive information about the trip plan. Section 2.3 compares the different kinds of mobile apps and gives a brief introduction of Android platform while this app is developed by Android Native app. Section 2.4 discusses Natural Language Generation that is the technology used to generate the trip plan summary in this project. Following that, Traveling Salesman Problem and possible solutions are discussed in section 2.5, along with the in-depth description of Genetic Algorithm which is an algorithm used to optimize the traveling route in this app. Finally, two related works: TripRec and Google Trips are discussed in section 2.6.

2.1 Web Services

Web Service is a network-addressable and platform independent software application that is available through a standard web protocol (HTTP or HTTPS). W3C defines web service as ”a software system designed to support interoperable machine-to-machine communication over a network”. [3] Other programs can communicate with the web service in XML or JSON which are the standard text-based formats can be easily recognized and parsed. According to the different architectural style, web services can be classified into two main categories:

- **Simple Object Access Protocol (SOAP):** SOAP is an object-oriented technology that has well-defined standards used for exchanging XML-based messages.

SOAP web service has to define its functionalities using Web Service Definition Language (WSDL) and wrap the result into a simple bulky SOAP message. Although SOAP supports loose coupling and distributed processing, it is message overhead and programming complexity.

- **Representational State Transfer (REST):** REST is a resource-oriented technology that consists of a set of design criteria that is implemented with Web standards (HTTP and URIs) based on uniform interface. Compare to SOAP service, RESTful service is much simpler and easier to use because it does not use interface definition like WSDL and the HTTP requests and responses do not contain any description overhead.

Tests made in [4] prove that RESTful services are more suitable for mobile applications because they provide smaller and faster responses due to its less overhead. The subsection will deepen into RESTful web services.

2.1.1 RESTful API

REST is an architecture style for designing loosely coupled applications over the Internet, which was introduced by Roy Fielding in 2000 [5]. REST defines six constraints applied to architecture [5]:

- **Uniform Interface:** Hypermedia As The Engine Of Application State (HATEOAS) is an important concept of RESTful architecture. Each resource should contain a hyperlink navigating to relative URIs to fetch related or additional information that enables the server to inform the client about how to change the state of the application.

- **Stateless:** there is no client context stored on the server between the request. Client is responsible for containing all the necessary information to perform the request, which is always contained in the URL, header or query parameter.

- **Cacheable:** the resources can be cached on the client or the server that improves performance and scalability through reducing the interaction between the client and the server.

- **Client-Server:** the client and the server are separated from each other. The server is focused on data storage and manipulation while the client is responsible for taking information and displaying it to the user. It improves the portability of the client code and scalability of the server.

- **Layered System:** the clients have no knowledge about what server or intermediate they are connecting. This helps components to be independent, enhance security, and improve the system scalability.
- **Code on Demand:** this is an optional constraint that the executable code may be downloaded to extend client functionality.

RESTful web services are web applications built upon the HTTP and REST architecture. They expose resources through web URIs, and use the HTTP methods: GET, POST, PUT or DELETE, which are mapped to CRUD actions: Create, Remove, Update and Delete [6]. RESTful API separates user interface involved from data storage that improves flexibility and loose coupling. The traveling data in the app will come from the following RESTful APIs:

- **TfGM API:** TfGM API [7] is developed by data.gov.uk, provides data from across the region's transport network in Greater Manchester, containing feeds for metrolink, car parks, accidents, etc. This data is real-time (2000 calls/minute) and semi-structured data which is provided as an Open Data Protocol Representational State Transfer Application Programming Interface (OData RESTful API) [8] that uses Hypertext Transfer Protocol (HTTP) requests to interact and the response content would be in JSON format. The example request URL is shown as follow:

  https://api.tfgm.com/odata/Metrolinks[?$expand][&$select][&$filter][&$orderby] [&$top][&$skip][&$count]

  The project will focus on retrieving the transport information, such as train, bus, or driving from this API to plan the route between attractions.

- **Google Places API [9]:** Google Places API is a RESTful API, provides places information in JSON or XML format by using HTTP requests (https:// protocol and API key). The requests include place search, place details, place add, place photos, place autocomplete and query autocomplete. The example request URL [10] is shown as follow:

  https://maps.googleapis.com/maps/api/place/nearbysearch/json?location=-33.8670522,151.1957362&radius=1500&type=restaurant&keyword=cruise&key=YOUR_API_KEY

The response of this request would be a list of restaurants which contain the word 'cruise' within a 1500m radius of a point in Sydney, Australia. This API will be used to query the suitable places base on user's preference and present additional information about each place as well.

- **Google Maps API [11]:** Google Maps is a web-based mapping service application and technology developed by Google that provides detailed information about geographical regions, street maps, aerial and satellite views around the world. The Google maps API allows developers to create a customized map by adding markers, polygons, and overlays to a basic map, and zoom in to a particular map area. This API will be used to create a map showing the trip plan route.

- **Sygic Travel API:** Sygic Travel [12] is a trip planner on the web and the mobile device that is the first online interactive maps designed for travelers. The maps are developed on OpenStreetMap and integrate with the Sygic GPS Navigation to guide the traveler. The place information comes from Sygic Travel editors, Wikipedia and other sources, including descriptions, opening hours, admission fees and so on. Sygic Travel provides an API that allows developers to retrieve place information, such as description, contact information, and expected duration of visit. Some data will be retrieved from this API to give the introduction of the attraction.

- **Foursquare API:** Foursquare [13] is a location-based social networking application for mobile phones. The users can use mobile devices to check-in to share their location and give some feedback about that place. Additionally, they can establish friend relationships to view each other's updates and find out when friends are nearby. Foursquare provides a places API that allows developers to retrieve place information and the feedback, such as total check-in number, average rating score, and visitor's reviews. The app will obtain some data from this API to give more information about the attraction.

2.1.2 Open Data Protocol

Open Data Protocol (OData) [8] is a set of open standards that allow data consumers and get data in a standardized way, which was initiated by Microsoft in 2007 and standardized at OASIS in 2014. Publishers of data services have to provide resource identifications, such as service document and metadata document, to help clients navigate the model in a hypermedia-driven fashion and understand the way to query and interact with entities. OData enables the creation and consumption of interoperable RESTful APIs, which allow clients to use HTTP protocol to publish and edit resources, and exchange data in XML or JSON format. The main feature of OData is data query, similar to SQL, it supports predicates, projections ,and aggregation. For example, *$select* is for projecting specified fields, *$skip* is for excluding first *N* entries in data, *$filter* is for selecting entries satisfying certain predicate expressions, and *$format* is for specifying return format (Atom, XML, JSON).

2.1.3 Data-Exchange Formats

XML and JSON are common data-interchange formats for web service, both of them are text-based open standard for representing data. In the following context, the pros and cons of each format will be discussed and figuring out which format is more suitable for the mobile application.

- **Extensible Markup Language (XML):** XML is an application profile of Standard Generalized Markup Language (SGML). It is a self-describing message format designed to store and transport data over the Internet, supports a wide variety of applications, and is widely used in a Services Oriented Architecture (SOA). There is also a multitude of XML-based formats available, such as RSS, Atom, and SVG. The two commonly used methods for parsing XML Documents are Document Object Model (DOM) and Simple APIs for XML (SAX); however, it is very expensive to implement them on the mobile platform.

- **JavaScript Object Notation (JSON):** JSON was derived from the European Computer Manufacturers Association (ECMAScript) programming language that is a text-based, language-independent and self-describing data interchange format. It is lightweight, easy for applications to parse and generate, and more efficient alternative to XML, as it does not have as much markup overhead. JSON presents data by simple key/value pairs and associative arrays. Despite its advantages, it has some shortcomings, such as its lack of namespace support and input validation, and it may not be as readable as XML or HTML.

Although both of XML and JSON are the great data-exchange formats, regarding the limit bandwidth for the mobile devices, choose an efficient way to present data is relevant important. Compare to XML, JSON is more lightweight and easier to parse. Furthermore, according to the test from [14], it proves that the use of JSON as data-exchange format is more efficient than XML due to the less consumption of bandwidth and computing time on parsing data.

2.2 Web Mashups

Web mashup is a popular technique that takes information from one or more sources and presents it in a unique user interface [15], which means it enables users to create situational applications based on existing application components. Recently, mashups have grown rapidly, one of the reasons is that many organizations, such as Google and governments, have opened up their data for the public to use. Moreover, the advent of new tools that make developers create mashups easily without knowing their technical knowledge. Web API is the key element of a web mashup, a popular API to be used in mashups is Google Maps. Through Google Maps API, developers can access the maps from Google and combine these maps with another stream of data to create a new application.

The benefit of mashups is that combining the multiple services enhances the usefulness and uniqueness of the application. Furthermore, reducing the effort to build and maintain many services while each web service is supported by their respective service providers. Besides the advantages, it also comes several disadvantages. A disadvantage of mashups is that the tight dependency with the involving web services, the user has no control over the quality or the features of them and there is no guarantee that this service will be continuously supported. Another drawback is that the contents used in the mashup service are not guaranteed to be secure, if there is the sensitive data contained in the application, incorporating the mashup API may pose a security concern. [16]

Trendsmap [17] is one example of the mashups that combines popular, location-specific Twitter topics with data from another Twitter trend site and displayed it on a map derived from Google Maps. Skyscanner [18] is another example of mashups that consumes various web services from different service providers to offer the price comparison service. Web mashup is an important technique used in this project to present the integrated trip plan.

2.3 Mobile App

There are three types of mobile applications, native app, hybrid app and web app. Each of them has its own benefits and targets, and these will be discussed in the following context, whilst Table 2.1 shows the comparison of these three apps. Furthermore, the Android platform will be introduced in the subsection. [19]

- **Native App:** Native app is developed for a particular platform and compiled directly on the device based on default language, which is iOS on Objective C or Swift, Android on Java, or Windows Phone on Net, and cannot be used on different platforms. The main benefit is high performance and good user experience as each platform providing specific features. The disadvantage is that it can only be developed for one platform by specific language and APIs at a time.

- **Web App:** Web app is a customized website that looks and feels like a native app. It is usually written in HTML5, JavaScript or CSS for client-side and possibly Java, PHP for server-side, running on a browser that users can access from any device whenever there is an internet connection. The advantages include minimum of device memory is required, and it can be accessible from any device which has a browser. The drawback is that developers can only use the APIs provided by browser rather than native APIs or the platform.

- **Hybrid App:** Hybrid app is a web app disguised in a native wrapper so that it can access to the features of the mobile device, such as camera or GPS. Hybrid app is compiled by a third-party tool, such as Apache Cordova, transforming the web app into a native app. The advantage is that it is fast and easy to develop because of single code base for all platforms. On the other hand, hybrid app has worse performance than native app and there may be some design issues for running on the different platforms.

	Native app	Hybrid app	Web app
Development cost	high	moderate	low
Performance	fast	moderate	slow
Distribution	App store		web
Device features	Wide access	Some access	Few access
Cross platforms	no	yes	
Maintenance	hard	easy	
Recommended for	Apps that will be developed for single platforms	Applications that need to be distributed as multi-platform	Applications with limited funds, resources or terms
	Apps with wide requirements due to capabilities of hybrid or web	Those apps that will be developed for App Stores	Apps that do not require App Stores
	Anything that require highly optimization level for stable work		Developed with HTML, CSS, JavaScript etc.
	Apps that need best native UI or best graphic animation		

Table 2.1: comparison of native, hybrid, web app

[20]

Considering the quality of user experience, performance and availability of native APIs, this app will be developed as a native app on Android platform.

2.3.1 Android Platform

Android is an open source, Linux-based software stack created for the mobile devices, which was founded by Android Inc. in 2003 and purchased by Google Inc. in 2005 [21]. The overall system architecture is found in Figure 2.1.

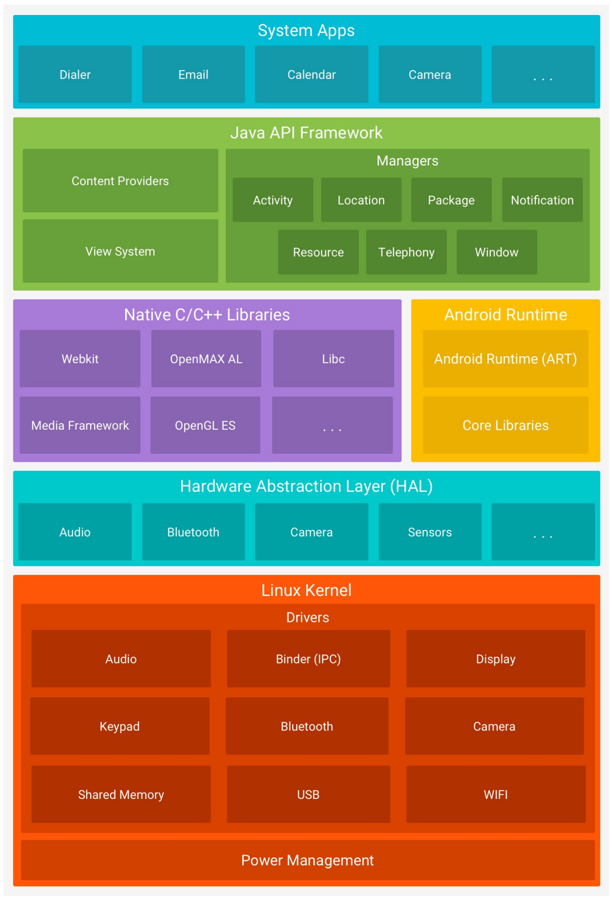


Figure 2.1: the Android platform architecture

[22]

- **The Linux Kernel:** Linux kernel is the foundation of the Android platform that the functionalities include low-level memory management, process management, networking and other OS related services.

- **Hardware Abstraction Layer (HAL):** HAL consists of multiple library modules and provides several standard interfaces for the framework API to access device hardware components, such as the camera or bluetooth.

- **Android Runtime (ART):** ART is written to run multiple virtual machines on low-memory devices by executing Dalvik Executable (.dex) files, which is optimized for the low memory requirements of mobile devices. The major features include Ahead-of-time (AOT) and just-in-time (JIT) compilation, optimized garbage collection (GC) and better debugging support.

- **Native Libraries:** The Android native libraries are written in C or C++ that can be accessed through the Java framework APIs. For example, accessing OpenGL for 2D and 3D graphics through the Android framework's Java OpenGL API.
- **Java API Framework:** This layer provides the high-level building blocks through APIs written in Java for the developers to create an Android application. For example, view system for building UI, resource manager for accessing to non-code resources, and activity manager for controlling the lifecycle of an application. Figure 2.2 shows the lifecycle of an activity.

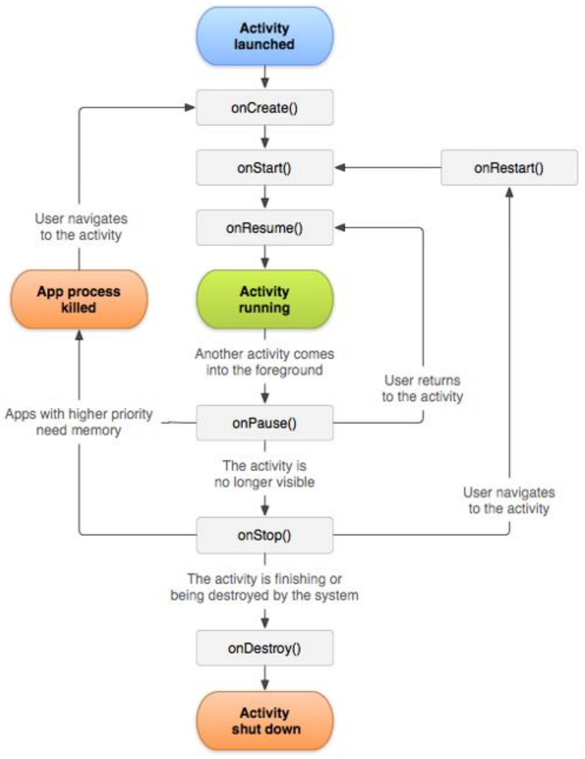

Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Figure 2.2: the lifecycle of an activity

[23]

- **System Apps:** This top layer includes a set of core applications for email, calendars, web browser, and more. These apps cannot only provide the functionalities for users but also allow developers to access from their own app.

The Android platform is free and available as open source software that enables devices to be customized without restrictions. Moreover, Android is based on Java programming language that is a well-known language has a broad support of adoption. Finally, there is a set of key applications available built from third-party developers and a wide range of free documental support is provided for all the developers. [24]

2.4 Natural Language Generation

Natural Language Generation (NLG) is a subtype of Artificial Intelligence (AI) that aims to generate natural language, such as summaries, reports, explanations, from a machine-readable format. The simplest form of NLG is translating data into text, like MS Word mail merge, retrieving some data from another source to fill in the gap. Another form is to use a template to display the output, which dynamic data is generated by predefined rules. The more complex form of NLG is to produce an insightful narrative by considering the context, domain, model etc. that users can comprehend. [25]

2.4.1 NLG Tasks

The entire process of NLG is complex and involves a number of criteria to render an output that looks natural; therefore, it has to be split into several tasks. The typical stages of natural language generation (Reiter & Dale, 1997, 2000) [26] are:

1. Content determination: Deciding which information to mention in the text.
2. Text structuring: Determining the overall hierarchy of information in the text.
3. Aggregation: Organizing the sentences to enhance the naturalness.
4. Lexicalization: Finding the appropriate words for the narrative.
5. Referring expression generation: Selecting the words and phrases to identify objects and regions.
6. Realization: Creating the suitable text according to the rules of grammar, morphology, etc.

2.4.2 NLG Tool

Realization is a subtask of NLG, which aims to create the actual words and phrases for the well-formed sentences by several rules. There are a number of open-source surface realizers available, including SimpleNLG, KPML, and OpenCCG. Table 2.2 shows a brief comparison of these tools, and SimpleNLG is used in this project, which will be explained in Section 2.4.4.

- SimpleNLG *(A. Gatt and E. Reiter., 2009)*[27] : a simple Java API which is intended to function as a ”realization engine”, focused on limited grammatical coverage compared to the other tools but easy to learn and use.
- KPML *(Bateman, 1997)*[28] : the oldest realizer written in ANSI Common Lisp with the graphical interface, which offers a mature platform for large-scale multilingual grammar development and generation.
- OpenCCG *(White, 2006)*[29] : an open-source realizer written in Java which provides many appropriate parsing and realization services according to Mark Steedman’s Combinatory Categorial Grammar (CCG) formalism.

	SimpleNLG	KPML	OpenCCG
Language	JAVA	ANSI Common Lisp	JAVA
Learning difficulty	easy	hard	
Advantage	Easy to learn and use	Mature platform with graphical interface	Provide various parsing and realization services
Disadvantage	Provide limited functionality	Developed by unfamiliar language	Less learning resources can be accessible
Focused on	Simple realization service	Large-scale multilingual development	Parsing and realization services

Table 2.2: comparison of NLG tools

2.4.3 NLG Evaluation

There are many ways to evaluate NLG systems, such as task-based, human rating, and metrics. Table 2.3 shows a brief comparison of these evaluations while human rating is used in this project, which will be explained in Chapter 5.

	Task-based	Human ratings	Metrics
Time and effort consuming	long	moderate	short
Advantage	Reflect the real-world effect	Broader insights about a system can be obtained	It is the quickest and cheapest way
Disadvantage	Sometimes it is difficult to evaluate the system in real-world	It may not evaluate a specific hypothesis	It may not reflect real-world effect
Example system	STOP (which produced smoking-cessation letters)	Babytalk BT-Nurse system (which generates nursing shift handover reports)	Translation system (which translate text or speech from one language to another)
Recommended for	When the time and effort is allowed, and a large number of subjects can be found	When it is impossible to carry out the task-based evaluation	When it is impossible to carry out the task-based evaluation or human ratings evaluation

Table 2.3: comparison of NLG evaluations

[30]

- **Task-based (extrinsic) evaluation:** is to try a system in the real world and see if it has the desired effect. This evaluation is time and effort consuming, and usually needs many subjects because there are lots of statistical noise, such as ethical challenges, technical issues.

- **Human ratings evaluation:** is to ask the subjects to use the system, and then give some feedbacks about the system. For NLG system, users are typically asked to rate the usefulness, accuracy, readability, and comments of the generated texts.

- **Metrics evaluation:** is to compare the generated texts against a set of reference texts by a specific metric, such as BLEU, METEOR, and ROUGE. This type of evaluation is broadly used in machine translation and document summarization.

2.4.4 SimpleNLG for project

According to the brief comparison of different NLG tools in Section 2.4.2, SimpleNLG is chosen to be used in this project. SimpleNLG is a realizer for a simple grammar, which can be used to generate English sentences with correct grammar. It is a Java library distributed as a jar file that includes multiple classes, allowing programmer to specify the subject, verb, object, and additional complements, also indicate the tense or the form of verbs. These classes perform the tasks, such as inserting appropriate whitespace in sentences and paragraphs, adding appropriate punctuations, formatting lists (i.e. A, B, and C), dealing with inflected forms (i.e. tense, gender, or number), and organizing all the different parts or phrases into the correct structure for English. Furthermore, most of the components in the library are self-contained modules so that it gives developers more freedom to use the library, allowing developers to extend or replace some parts with their own code. [30]

2.5 Traveling Salesman Problem

TSP is about a salesman and a set of cities that the salesman plans a trip to visit each of his clients (cities) from a certain point and returns to the same point. During the trip, every client should be exactly visited once. The challenge of the problem is that the route should be minimized the total length and less costly. TSP can be modeled as an undirected weighted graph: let $G = (V, E)$ be a graph that V is a set of vertices, E is a set of edges, and let $C : (C_{ij})$ be a distance matrix associated with E. The purpose is finding a minimum-weight Hamilton circuit (a circuit that visits each vertex exactly once) in a complete graph (each pair of vertices is connected by an edge). [31]

2.5.1 Formulation

TSP was formulated as an integer linear program proposed by Miller, Tucker and Zemlin (1960) [32]. Firstly, the variable is defined:

$$x_{ij} = \begin{cases} 1 & \text{if the edge } i \rightarrow j \text{ is in the tour} \\ 0 & \text{otherwise} \end{cases}$$

Then, the problem is defined as follows:

$$min \sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} c_{ij} x_{ij} :$$

$$0 \leq x_{ij} \leq 1 \qquad i, j = 1, ..., n;$$

Subject to:

$$\sum_{i=1, i \neq j}^{n} x_{ij} = 1 \qquad j = 1, ..., n; \qquad (1)$$

$$\sum_{j=1, j \neq i}^{n} x_{ij} = 1 \qquad i = 1, ..., n; \qquad (2)$$

$$x_{ij} \in \{0, 1\}, \forall (i, j) \in A$$

Where (1) means that each city has to be connected from exactly one other city, and (2) means that each city has to link to exactly one other city. In addition to the above usual assignment constraints, in order to prevent subtours, the following subtour elimination constraints (SECs) are defined to degenerate tours that are formed between intermediate nodes without connecting to the origin. It means that there is only a single tour rather than many disjointed tours to cover all cities.

$$u_i \in Z \qquad i = 1, ..., n;$$

$$u_i - u_j + n x_{ij} \leq n - 1 \qquad 2 \leq i \neq j \leq n;$$

$$0 \leq u_i \leq n - 1 \qquad 2 \leq i \neq j \leq n.$$

2.5.2 Possible Solutions for TSP

The most direct solution for a TSP would be brute-force approach that enumerates all the possible tours and see which one is cheapest $(O(n!))$. This solution is impractical and time consuming, finding an efficient solution is necessary. The solutions are divided into two types: exact and heuristic. The exact algorithms are suitable for small problem size that the optimal tour is available. The heuristic algorithms quickly provide good solutions; however, the optimal route is not guaranteed.

- **Branch and bound (B&B) algorithm:** B&B is one of the exact algorithms that is proposed by A. H. Land and A. G. Doig in 1960 [33]. It is a popular tool for solving NP-hard combinatorial problems through enumerating and pruning the search space by using upper and lower bounds. This algorithm consists of two parts: the first part is branching that splits a set of candidates (S) into several subsets $(S1, S2, ...)$, and define a rooted tree which nodes are the subsets of S. Another part is bounding that set upper and lower bounds within a given subset. The next step is a recursion called pruning, any node whose lower bound is greater than the minimum upper bound would be discarded while it stops when the current candidate set is reduced to a single element or the upper bound matches the lower bound. By implementing it, the time consumption for the computation is reduced since a lot of possible routes are not taken into account. However, the complexity of this algorithm increases with the number of places to visit. [34]

- **Nearest neighbor (NN) algorithm:** NN is a heuristic solution for TSP that the salesman starts at a random city and goes next to the nearest unvisited city until all have been visited. The running time of it is $\theta(N2)$ [35] and implemented as the following steps:

  1. pick an arbitrary vertex as starting vertex.
  2. find the nearest unvisited vertex for the current vertex and set it as current.
  3. repeat step 2
  4. terminate the recursion if Hamilton cycle is complete (all the vertices are visited)
  5. repeat steps 1 to 4 until all the vertices have been picked up as starting vertex
  6. pick the Hamilton cycle which has the shortest distance as the result

  NN algorithm is a quick and easy method to solve TSP; however, the result may not be the optimal tour.

2.5.3 Genetic Algorithm (GA)

GA is offered as the solution for TSP and determine the optimum route on google map in this project. GA is a heuristic search algorithm based on the evolutionary technique that was invented by John Holland in the early 1970's [36]. According to Charles Darwin's survival of the fittest evolutionary theory, only the fittest individuals in a population are likely to survive and produce offspring. Through this principle, the search result is optimized by using historical information. GA begins with generating an arbitrary population as an initial generation, each individual represents as a possible solution to the problem whilst each of them will be evaluated by the fitness function to determine how fit they are. After that, the algorithm implements the evolutionary process through three main operators [37]:

- **Selection:** two pairs of individuals with the highest fitness are selected as parents and their genes will be passed to the next generation.

- **Crossover:** this is the most important operator in GA that the offspring are produced by exchanging the genes among parents depends on the randomly chosen crossover point. For example, if $p1 = 000000$ and $p2 = 111111$ and the crossover point is 2 then $p1' = 110000$ and $p2' = 001111$. The new offspring created from this mating are added to the population.

- **Mutation:** some of the new offspring will have some of their genes flipped; for example, $s1 = 110000$ mutate to $s1' = 011000$. The purpose of it is to prevent premature convergence and maintain diversity within the population.

The algorithm repeats the above process until the fitness individual is good enough. Figure 2.3 shows the flowchart of a Genetic Algorithm.

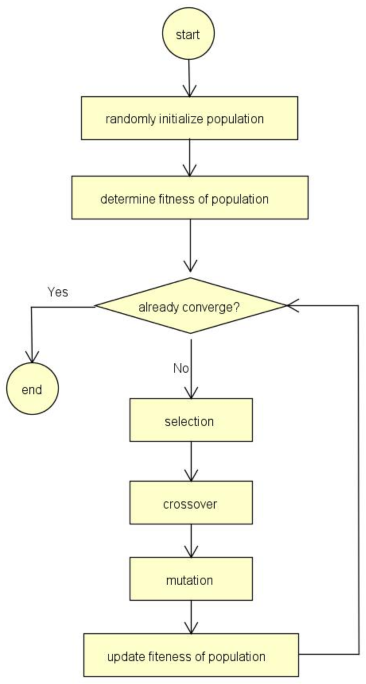

start
randomly initialize population
determine fitness of population
already converge?
Yes
end
No
selection
crossover
mutation
update fiteness of population

Figure 2.3: flowchart of Genetic Algorithm

[38]

2.6 Related Works

The two applications, TripRec and Google Trips, are related to this project that will be introduced in the following context. Both of them are the trip recommenders about automatically generating a trip plan based on the user's preference.

- **TripRec [39]:** this paper proposed a recommendation system that combines the knowledge-based and community-based approaches to plan a trip depends on the user's preference. The system starts at interacting with a group of pictures that represent different types of tourist destinations and selecting the time budget for traveling. Following that, an one day trip will be created by calculating user's interest, location, and number of checkins on Foursquare. However, it is developed as a web application rather than providing a mobile application.

- **Google Trips [40]:** this mobile application is provided by Google Inc. that enables users to store the travel information, discover the things to do, and make a trip plan. One of the functionalities, Day Plans, is related to this project. It automatically suggests a half day or a full day trip on the map and allows users to customize a trip based on their interests and available time. However, the user's input is simply focused on traveling time, such as the day of the week and half day/full day, without other conditions, like the preferred transport or the types of attraction.

Chapter 3

Methodology and Application Design

This chapter presents the methodologies and designs for achieving the goals of the project, it begins by introducing the agile development methods like XP and Scrum, then providing an appropriate process for developing this project. Section 3.2 gives a description of the MVC pattern which is the structure used in this project to build the application. Section 3.3 lists the functional and non-functional requirements with complexity and priority while Section 3.4 shows the considerations for automatically generating a trip plan. Section 3.5 illustrates the system components as a bottom-up structure and the overview of the implementation process is presented in Section 3.6.

3.1 Agile Development Methodology

Agile development methods which follow the principle defined in Agile Manifesto, have become popular in recent years. This type of methodology is focused on frequent communication, delivering working software, interacting with customer, and responding to change, which is suitable for the project that the requirements may need to be changed frequently. [41] Typically, Agile process consists of several fix-length repeated cycles which include requirement analysis, planning, implementation, and testing. The aim is to improve productivity, enhance the quality of system, and meet user's requirements. There are many different kinds of agile methodologies, such as lean and kanban, scrum, extreme programming, dynamic systems development method [42], two of them are introduced as follow:

1. **Extreme programming (XP):** XP is a lightweight methodology suitable for small size projects, which is originally proposed by Kent Beck in 1999 [43].

The aim of XP is to deliver high-quality software quickly and continuously that the features include customer involvement, rapid feedback, continuous testing and planning, and delivering working software frequently. XP is based on four activities: coding, testing, listening, designing, along with five values: communication, simplicity, feedback, courage, respect, and twelve practices [44]:

- **Planning game (User stories):** discuss the system requirements with the customers and define a list of desired features as user stories. Each story contains a brief description of user valued functionality, prioritization, and estimated effort consumption. These user stories are used for cost estimating and project management.
- **Small releases:** the whole developing process consists of numerous short cycles. In each cycle, a small set of features are added and a working software is released.
- **On-site customer (Customer acceptance tests):** the customers should be available at all time to guide the development, set priorities for the tasks, and answer questions. This ensures that the developers can obtain the more accurate requirements and quick feedback.
- **Simple design:** it follows the YAGNI (You Aren't Gonna Need It) principle that a feature should be implemented only when it is clearly required without considering future requirements. It keeps the clean code and the fewest possible classes.
- **Pair programming:** two programmers work on a piece of code that one person writes the code while the other reviews it, and then switch the roles. This improves the quality of the code in an efficient way through writing and reviewing the code at the same time.
- **Testing (TDD):** XP adopts test-driven development (TDD) as part of software development process that the test is written before implementing the code. Generally, XP contains two types of tests, one is unit test which is automated test to test a single or a small cluster of classes by developer. Another one is acceptance test that usually tests the overall system or a large chunk of it by customer. Passing all the acceptance tests is the criteria for the complete work. The detailed introduction of TDD will be described in chapter 5.

- **Refactoring (Design improvement):** this is the process for improving the existing codebase without changing behavior or adding any functionality. It is the time to remove duplicate code and make it more understandable.

- **Continuous integration:** integrate the new code with the main codebase frequently and ensure all the tests pass. This prevents the diversion and makes sure the newest system is available for all developers.

- **Collective code ownership:** all the code belongs to all the team members, which means everyone can refine any code at any time. This encourages everyone to share their knowledge to make the system better.

- **Coding standards:** everyone writes the code in the same style and format that benefits for code sharing and integration.

- **Metaphor:** standardize the system of names that each name should be intuitive and easy to remember. This allows programmer to understand the functionality through its name and easily communicate with other members.

- **Sustainable pace (Forty-hour workweek):** make sure developers not working overtime (normally forty hours per week). This maintains the quality of code and the level of productivity.

Don Wells [45] depicted the XP process and the planning/feedback loops as diagrams which are showed in Figure 3.1 and Figure 3.2.

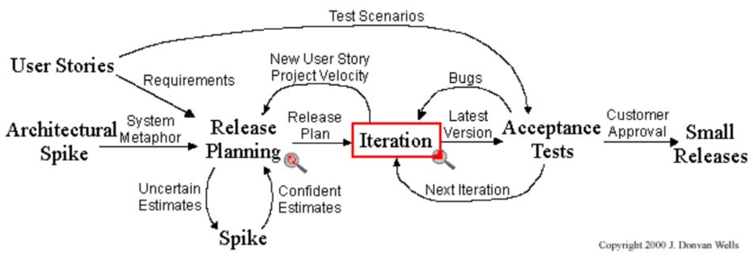


Figure 3.1: XP process derived from

[46]

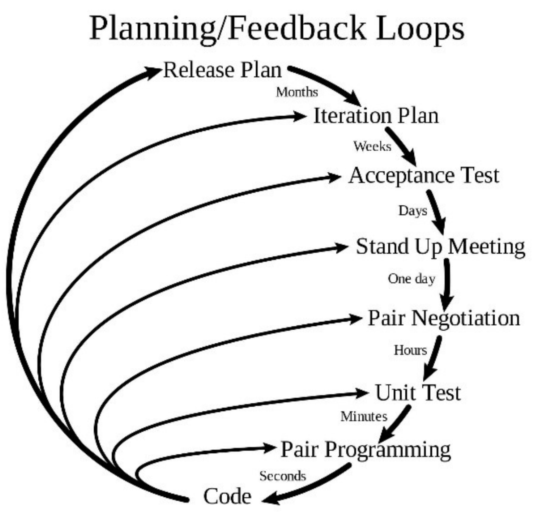


Figure 3.2: planning/feedback loops derived from [47]

2. **Scrum:** a lightweight framework for project management utilizing iterative and incremental approaches, which was introduced by Hirotaka Takeuchi and Ikujiro Nonaka in 1986 [48]. This type of methodology is designed for responding to change and quick delivering a working software. Moreover, it is suitable for large project which may consist of multiple teams. Scrum is defined by team roles, events, and artifacts. There are three main roles in a Scrum team. The Product Owner is the stakeholder of the project who is responsible for defining and making priority of the features, accepting or rejecting work results, and communicating with team and other stakeholders. The ScrumMaster is responsible for enacting Scrum values and practices include removing impediments, ensuring the team is functional and productive, facilitating meetings, and so on. The Development Team is a self-organizing and cross-functional group which may include business analyst, architect, developer, or user experience specialist. There is no hierarchy in the Development Team. [49] The Scrum workflow contains several events [49]:

   - **Sprint planning:** it happens at the beginning of a sprint to determine the tasks and set the goals for the following Sprint.
   - **Sprint:** a time-box period of development that can be from one week to one

month, but typically two weeks. A sprint has a common set of following activities.

- **Daily scrum:** it is also known as Daily stand-up which is a short meeting that each team member briefly describes the progress, planned work, and impediments.
- **Sprint review:** it is the time to present the result of the Sprint, typically takes the form of a demo of new features or underlying architecture. The Product Owner decides whether the work is acceptable, and the clients give feedback to ensure that the system meets the requirement.
- **Sprint retrospective:** it happens after Sprint review that the whole team gathers to discuss what features would start doing, stop doing, and continue doing. It is an opportunity for the team to identify the performance during the Sprint and find the strategy to improve the process.

The Scrum artifacts include [49]:

- Product backlog: a list of requirements for the project, each item has value to the customers. All the items are prioritized by the Product Owner and reprioritized at the start of each Sprint.
- Sprint backlog: a small set of items split from the product backlog which will be implemented in a Sprint.
- Product increment: in each Sprint, the Development Team produces a potentially shippable increment that describes the sum of complete works in the product backlog. This increment must be acceptable by the Product Owner and all the items included in it must be ready to be released.

There are also some optional artifacts like burn-down charts and task boards which are commonly used in Scrum team. Figure 3.3 shows the process of Scrum.

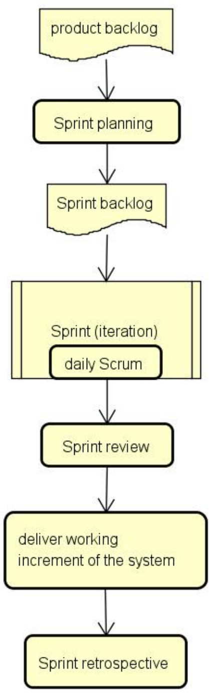


Figure 3.3: Scrum process

3.1.1 Agile Development Methodology for Project

This project is developed with a single developer and with a short time frame; therefore, the use of a specific agile methodology is considered to be inappropriate since the most of agile methodologies like XP and Scrum are focused on team interactions. Although some practices like pair programming and multiple roles in a project are not suitable for this project, some of the principles are helpful for constructing an efficient development process, such as iterations, TDD, and artifacts. This project integrates some parts of principles from XP and Scrum for development process. The development process starts with creating a product backlog which contains functional and non-functional requirements, and then split them into Sprint backlogs during Sprint planning. The goal of each Sprint is to complete the tasks in Sprint backlog through TDD process, and the Sprint performance is evaluated by a burn-up chart. The following stage is review and retrospective that the Sprint result is demonstrated to the supervisor (Product Owner) and the product backlog is refined, a new Sprint starts afterward. Figure 3.4 shows the development process.

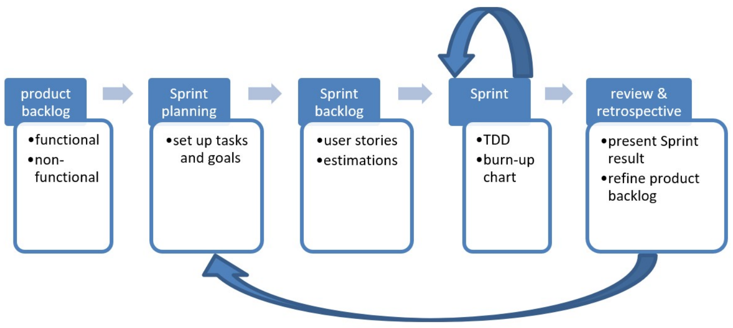


Figure 3.4: development process

3.2 MVC Pattern

Model-View-Controller (MVC) is a design pattern introduced by Trygve Reenskaug in 1970s at the Xerox Parc. [50] MVC pattern separates the responsibilities for an application into three main components [51]:

- **Model:** a data layer that is responsible for managing the business logic and the state of the application.
- **View:** a UI layer that is responsible for application design, split from application logic.
- **Controller:** a logic layer that is a mediator to connect model and view, responsible for handling user's behavior and updating the model.

MVC structure provides several benefits include developing high cohesion and low coupling software, enhancing testability of the code like unit test, and easier to extend. [51] Figure 3.5 shows the interactions within the MVC pattern.

This project is developed in MVC structure that each module is defined in a specific package. The Model module consists of several Java classes that manage the data, state, and business logic, such as global state, place and transport information, Genetic Algorithm, JSON parser, etc. The Controller module contains multiple Activity classes that handle the user actions from View and interact with Model; for example, when user clicks ”generate plan” button on View, Controller will go through the logic from Model by onClick event and then present the plan result on View. The View module involves several XML files defining the application layouts and menus, which may include buttons, text boxes, lists, and so on. Figure 3.6 shows the Model, Controller, and View module in the project.

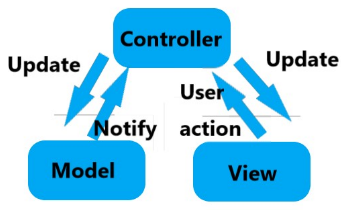
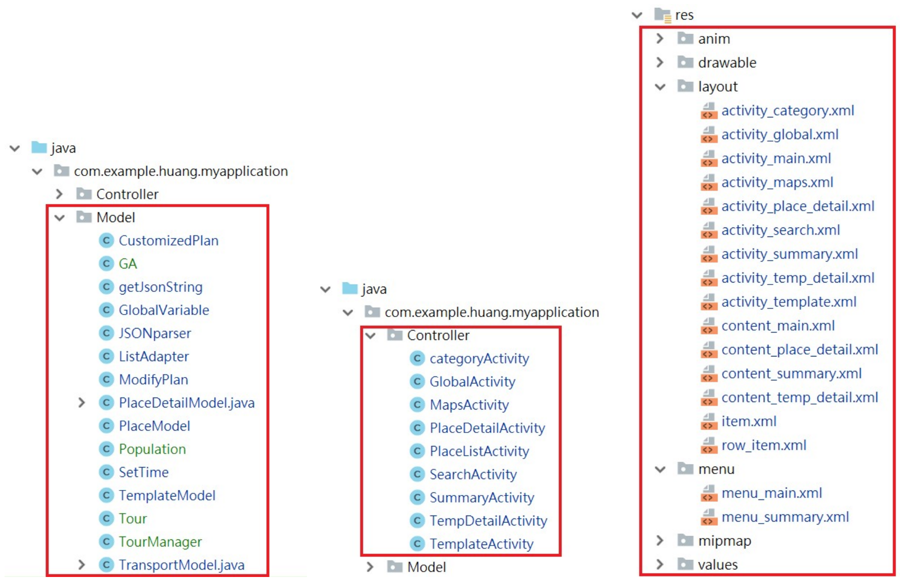


Figure 3.5: interactions within MVC



```
java
  com.example.huang.myapplication
    Controller
    Model
      CustomizedPlan
      GA
      getJsonString
      GlobalVariable
      JSONparser
      ListAdapter
      ModifyPlan
      PlaceDetailModel.java
      PlaceModel
      Population
      SetTime
      TemplateModel
      Tour
      TourManager
      TransportModel.java
```

```
java
  com.example.huang.myapplication
    Controller
      categoryActivity
      GlobalActivity
      MapsActivity
      PlaceDetailActivity
      PlaceListActivity
      SearchActivity
      SummaryActivity
      TempDetailActivity
      TemplateActivity
    Model
```

```
res
  anim
  drawable
  layout
    activity_category.xml
    activity_global.xml
    activity_main.xml
    activity_maps.xml
    activity_place_detail.xml
    activity_search.xml
    activity_summary.xml
    activity_temp_detail.xml
    activity_template.xml
    content_main.xml
    content_place_detail.xml
    content_summary.xml
    content_temp_detail.xml
    item.xml
    row_item.xml
  menu
    menu_main.xml
    menu_summary.xml
  mipmap
  values
```

Figure 3.6: Model, Controller, View module

3.3 Requirements

The first step of development process is establishing a product backlog that gathers requirements from the customers, each requirement consists of a general description for an actor and the expected performance of the system. The requirements are divided into two main categories, functional and non-functional. The functional requirements, also known as behavioral requirements, are related to the functionalities that the system must be able to complete, such as calculations, technical details, or data manipulation. [52] The non-functional requirements, also known as quality requirements, are related to the constraints that are imposed on the design or implementation, such as performance, security, and reliability. [53] These requirements not only improve the quality of application, but also enhance the user experience which is important for designing a mobile application. In order to plan the appropriate Sprints, each requirement is also classified by the degree of complexity and priority within low, medium, and high. Table 3.1 and Table 3.2 show the summary of functional and non-functional requirements with the complexity and the priority.

	Description	Complexity	Priority
1	Setting up of development environment	Medium	High
2	User must be able to get trip plan from template	Low	Medium
3	User must be able to choose different categories of the template	Low	Medium
4	User must be able to view a list of plans in each template category	Low	Medium
5	User must be able to get trip plan based on preferences	High	High
6	User must be able to set up multiple preferences (e.g. location, time, transport type, place type, etc.)	Medium	High
7	User must be able to view trip plan in a simple list	Low	Medium
8	User must be able to view trip plan in detailed summary	High	High
9	User must be able to re-generate a new trip plan	Low	High
10	The trip plan route must be optimized	High	High
11	User must be able to modify trip plan by deleting and adding places	Medium	Medium
12	User must be able to view trip plan route on map	High	High
13	User must be able to view the detailed information for each place (e.g. address, opening hour, rating, description, etc.)	Medium	High
14	User must be able to view place detail from different sources	Medium	Medium
15	User must be able to view a list of places not in trip plan	Low	Medium

Table 3.1: Summary of functional requirements

	Description	Complexity	Priority
1	Design interface should be simple, clear, easy to use	Medium	Medium
2	Design interface should be visual aids to facilitate its use (e.g. icons, colors, shapes, etc.)	Medium	Medium
3	The terms used in design interface components (e.g. labels, buttons, tables, etc.) should be consistent among all the application and match the users' vocabulary	Medium	Medium
4	Design interface should display trip plan results in a complete but easy to understand manner	Medium	Medium
5	Design interface should offer appropriate visualization modes for different functionalities (e.g. lists, tables, maps, etc.)	Medium	Medium
6	The application should be able to run on Android devices	Medium	High
7	The application should be easy and intuitive to use	Medium	Medium
8	The application should be simple and appropriate for travelers who travel in Greater Manchester	Medium	Medium
9	Trip route should be clearly shown on the map	Medium	Medium
10	The application should notify user when errors occur using design interface	Medium	High
11	The application should validate data entered by user to prevent errors	Medium	High
12	The application should provide enough information for traveling in Greater Manchester	Medium	Medium
13	The application should generate a reasonable trip plan	High	Medium
14	The context in the application should be fluency, accurate, easy to understand	Medium	Medium

Table 3.2: Summary of non-functional requirements

3.4 Considerations for Trip Plan Design

Making a good trip plan is time consuming because there are many things should be considered, such as sequence of the route, range of the traveling area, spot selection, staying time estimation, and so on. The main purpose of this project is to generate a trip plan automatically and one of the goals is to make the trip plan reasonable and executable, in order to achieve this goal, several constraints are defined to guide the planning process. Table 3.3 shows the summary of restrictions with reason and handling.

3.5 System Structure

The whole application is built from the multiple components which can be composed as a bottom-up structure that is divided into four layers. The bottom layer is data sources that include five APIs where the trip plan information comes from, and two JSON files that define the plan templates and NLG elements. The second layer from bottom is data processing that contains different libraries and algorithms to handle the data from the bottom layer, such as parsing JSON or generating summary. The third layer from bottom is Android framework that consists of multiple Java APIs in Android platform, which is the main process for developing an Android application. The view system provides the building block for UI components; the activity manager handles the interactions within activities, services, and processes; the resource manager is responsible for additional files like bitmaps, layout definitions, animation instructions, etc.; and the location manager allows to access location services, such as getting the permission for accessing device's location data or obtaining the updates of the device's geographical location. [54] Finally, the top layer is the trip plan application itself. Figure 3.7 illustrates the project bottom-up structure.

Constraint	Reason	Handling
All the places (e.g. start point, end point, attractions, etc.) must be in Greater Manchester	This app presents transport information by TfGM API that only provides transport for Greater Manchester	If any place is out of Greater Manchester, show an alert message on the screen
The end time of the trip must be greater than the start time	In order to get a valid traveling duration	If the end time is lesser than the start time, show an alert message on the screen
Trip duration must be longer than one-and-half hour	In order to remain enough time for visiting attractions and taking transport	If duration is shorter than one-and-half hour, show an alert message on the screen
The place-searching range depends on transport type	Different transport type is suitable for different distance of spots (e.g. cars for far distance, walking for near distance)	The radius from start point is based on different transport type (bus for 10 km, rail and driving for 30 km, walking and bicycling for 5 km)
In trip plan, choose one place per one-and-half hour	In order to remain enough time for visiting attractions and taking transport	Spot selection is implemented through a specific algorithm
The staying time for each spot is estimated as one hour	The staying time for each place is difficult to define; therefore, this app assumes that average visiting time is one hour	The visiting time is defined in an algorithm
The transport must be available	Transport is one of the key elements for a trip plan	If selected transport is not available, change it to walking that is always executable (e.g. if user prefer taking bus but it is no service to the destination, then walking method would be selected)
The route of trip plan is always optimized	Traveling by optimized route can save lots of time	Whenever the trip plan is modified (e.g. add or remove place), the route would be re-optimized
The route is optimized by traveling time	Optimize route by traveling time is more practical than distance	this project uses modified Genetic Algorithm (based on traveling time) for route optimization

Table 3.3: Summary of constraints

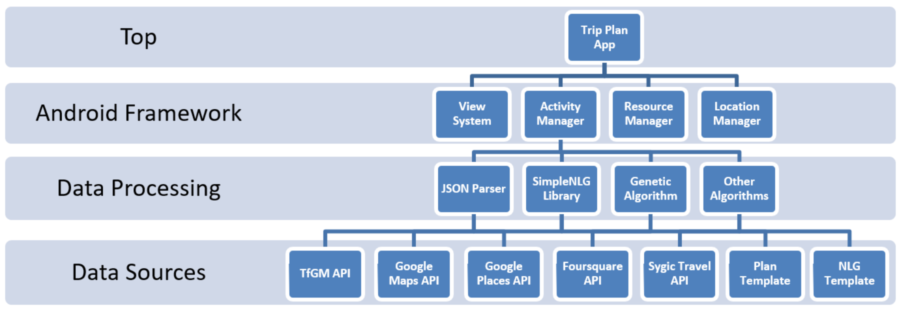


Figure 3.7: Project Structure

3.6 System Implementation Process

According to the above sections, the system requirements and the constraints have been determined in the Section 3.3 and Section 3.4 while the system structure has been presented in the Section 3.5. After that, the system implementation process should be considered so that each Sprint can be appropriately planned. The implementation process starts with setting up development environment that is the basic step to build an application, including IDE installation, project establishment, and so on. Following that, based on the priority of the requirements and the level of the system components, retrieving data from multiple sources is implemented because data is the urgent element for this application that the whole project is focused on how to process these data and present it in an acceptable way. After obtaining the data, data processing and data mashup are executed, which are the most complicated and important stage for this project. For processing data, several algorithms and techniques like GA and NLG are involved to achieve the goal. Otherwise, in data mashup, TfGM API would combine with Google Places API to present the trip plan summary while Google Places API would integrate with Google Maps API to show the trip plan on a map, and Google Places API would merge with Sygic Travel API and Foursquare API to provide the details for a specific place. The further step is to present information in design interfaces with suitable manner and the user experience should be also considered at the same time. The final step is to refine the application, such as improving the performance, complying with additional functionalities, enhancing user experience, and more. Figure 3.8 shows the overview of the system implementation process and it will be split into details and described in Chapter 4.

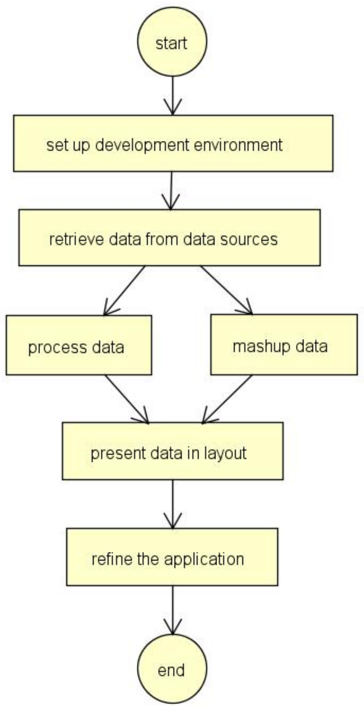


Figure 3.8: implementation process overview

Chapter 4

Application Implementation

This chapter presents the implementation process and result of the application, it begins by introducing the development environment like programming language, IDE, etc., then illustrating the overall state of the application through component diagram and activity diagram in Section 4.2. According to the agile methodology, the development process is divided into five Sprints that each Sprint lasts for two weeks from May to the middle of July. Section 4.3 to 4.7 describe the tasks and implementation results for every Sprint with backlogs, screenshots, algorithms, and a burn-up chart to show the performance.

4.1 Development Environment

This section discusses the selection of tools and technical support library that made the development environment of this project, and the reasons for the selection will be explained in detail.

- **Programming Language:** this application was implemented using Java as the main programming language that is a familiar language for the programmer. Java is a high-level, general purpose, concurrent computer programming language that offers concurrency and adheres to the object-oriented design paradigm through the use of classes. Besides, another reason for choosing Java is to fit the Android platform and SimpleNLG library that are written in Java language.

- **Android Version:** the minimum supported Android version for this project is API level 15 of the Android 4.0.3 (IceCreamSandwich) that allows the application to run on approximately all the devices with Android operating system. [55]

- **Integrated Development Environment (IDE):** Android Studio is applied as an integrated development tool in this project. Android Studio is Android's official IDE provides a set of instruments to help improve the efficiency when developing the application for Android device. It offers a rich code editing, debugging plugs, unit testing, performance tooling and support instant run, sample apps, layout editing, etc.

- **Technical support library:** To produce an expected summary for trip plan, SimpleNLG is selected to implement it, which is a Java API to facilitate the generation of Natural Language. The current released version is 4.4.8 that supports general lexicon, NIH specialist lexicon, sentence generation, and so on.

- **Version control tool:** this project manages the code versions through Gitlab that is a self-hosted Git-repository management system, which gives user complete control over the project and allows user to change access permission for free.

4.2 Overall Structure and Process

Before applying the exhaustive implementations, the general structure and process of entire system is considered in this section. There are two Unified Modeling Language (UML) diagrams will be introduced in the following context. First of all, a component diagram is depicted to represent the relationships between components. Furthermore, an activity diagram is provided to explain the operation flow of the application.

- **Component Diagram** Component diagram is a structural diagram that can be used to model the static implementation view of a system. These diagrams provide the visualization of the impending implementation, and describe the organization and relationships of the components. They describe the components used to make the functionalities rather than functionalities themselves. [56] A well-designed component diagram is able to improve performance, maintainability, and extensibility of a system. According to the Figure 3.7, the hierarchy of the components is illustrated and Figure 4.1 will present the interactions between some major components.

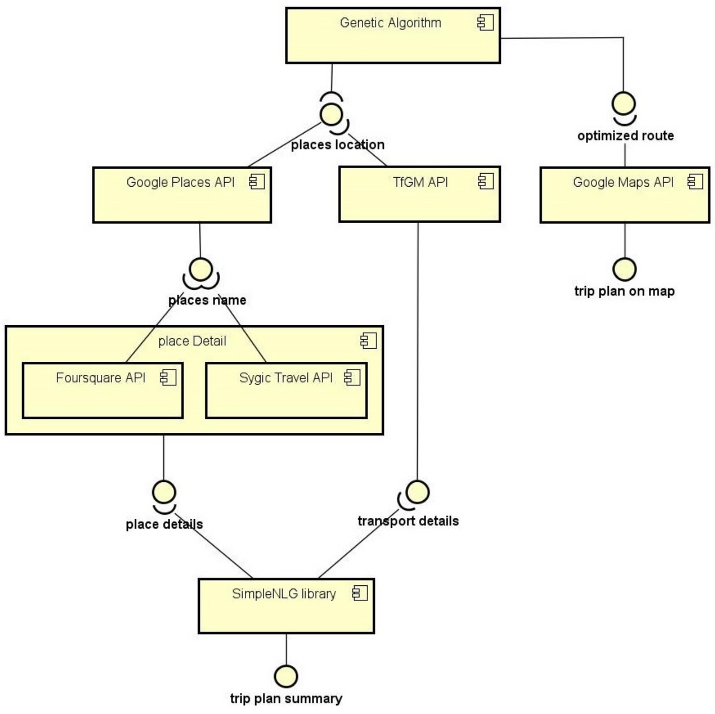


Figure 4.1: Component Diagram

Through the above component diagram, Google Places API is exploited to provide a set of places and their information like name and geographical location, then TfGM API retrieves place locations to offer transport information like traveling time and route details. Moreover, Genetic Algorithm also needs place locations to generate an optimized route which is requested by Google Maps API to show the trip plan on a map. On the other hand, Foursquare API and Sygic Travel API require place name which is provided by Google Places API, to query the further place information, such as telephone number and visitor reviews. Finally, SimpleNLG library requests the place details and the transport details which are provided by the previously mentioned components, to generate a trip plan summary.

- **Activity Diagram** Activity diagram is a behavioral diagram that can be used to model the dynamic aspects of a system. These diagrams provide a flowchart to express the flow from one operation to another operation that explains how activities are coordinated to provide a service. The flow control can be described as sequential, branched, or concurrent by using different elements, such as fork and join. [57] Figure 4.2 presents the overall activity flow of the application.

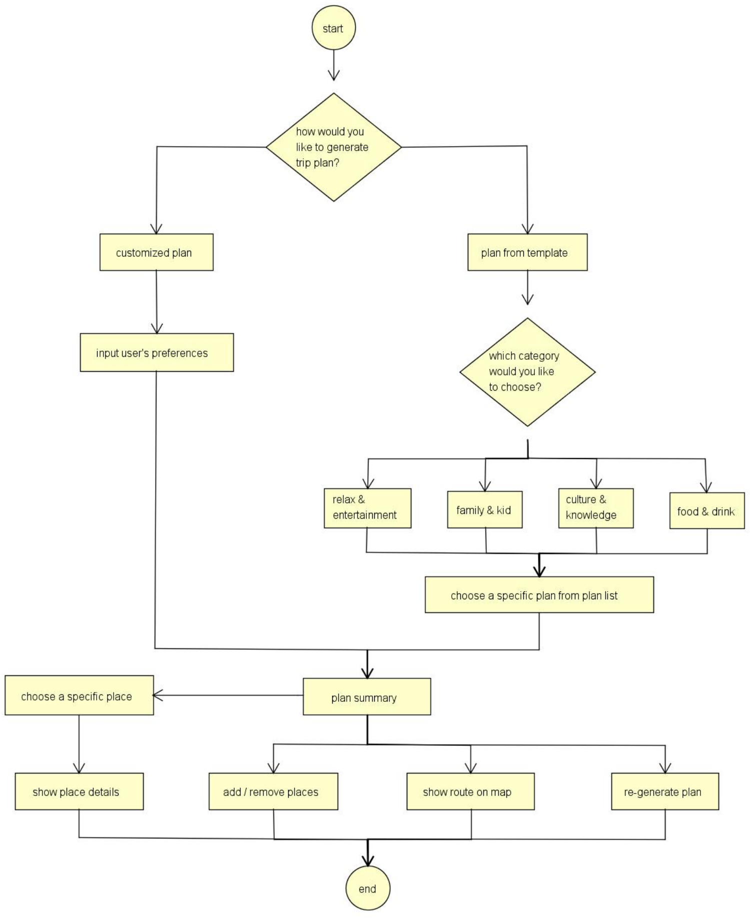


Figure 4.2: Activity Diagram

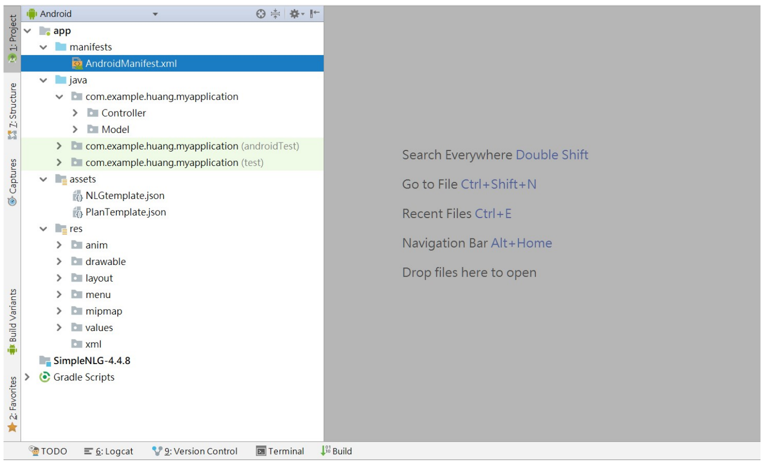
According to the above activity diagram, the process starts with deciding which way to generate a trip plan, through user's preference or from template. If user chooses customized plan, a trip plan summary will be created after selecting preferred time, location, transport, and so on. On the other hand, if user chooses plan from template, a specific category should be selected from four different types, and then a list of trip plan suggestions based on that category will be provided. User can further choose one of plans to obtain the summary of it. Following that, there are several activities can be operated, such as browsing the detailed information for a particular attraction, modifying the trip plan by adding or removing places, presenting the trip plan route on a map, and re-generating a new trip plan.

4.3 Sprint 1

In the first Sprint, the most important task is to determine all the environment has been set, such as installing IDE, creating an Android project, setting up the version control tool, etc., so that the application can be well-built. The aim of it is to make sure the application can run on Android devices. After that, some simpler functionalities are implemented, which are about establishing trip plan based on templates. Trip plan templates are stored in a JSON file so that a JSON parser should be implemented to explore the data that users need. Table 4.1 shows the Sprint backlog for Sprint 1. Figure 4.3 presents the project built in Android studio and Figure 4.4 provides part of commit history on Gitlab. The implementation results are also explained as follow while the JSON file of trip template will be illustrated in Appendix A.

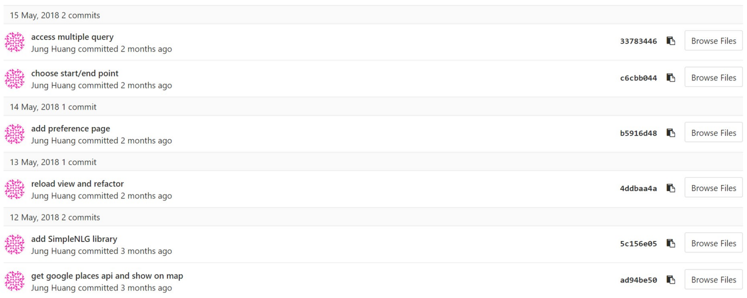
Type	User Story	Complexity	Priority
Functional	Setting up of development environment	Medium	High
	User must be able to get trip plan from template	Low	Medium
	User must be able to choose different categories of the template	Low	Medium
	User must be able to view a list of plans in each template category	Low	Medium
Non-functional	The application should be able to run on Android devices	Medium	High

Table 4.1: Sprint 1 backlog



```
app
  manifests
    AndroidManifest.xml
  java
    com.example.huang.myapplication
      Controller
      Model
    com.example.huang.myapplication (androidTest)
    com.example.huang.myapplication (test)
  assets
    NLGtemplate.json
    PlanTemplate.json
  res
    anim
    drawable
    layout
    menu
    mipmap
    values
    xml
  SimpleNLG-4.4.8
Gradle Scripts
```

Figure 4.3: project in Android Studio



15 May, 2018 2 commits

access multiple query
Jung Huang committed 2 months ago — 33783446 — Browse Files

choose start/end point
Jung Huang committed 2 months ago — c6cbb044 — Browse Files

14 May, 2018 1 commit

add preference page
Jung Huang committed 2 months ago — b5916d48 — Browse Files

13 May, 2018 1 commit

reload view and refactor
Jung Huang committed 2 months ago — 4ddbaa4a — Browse Files

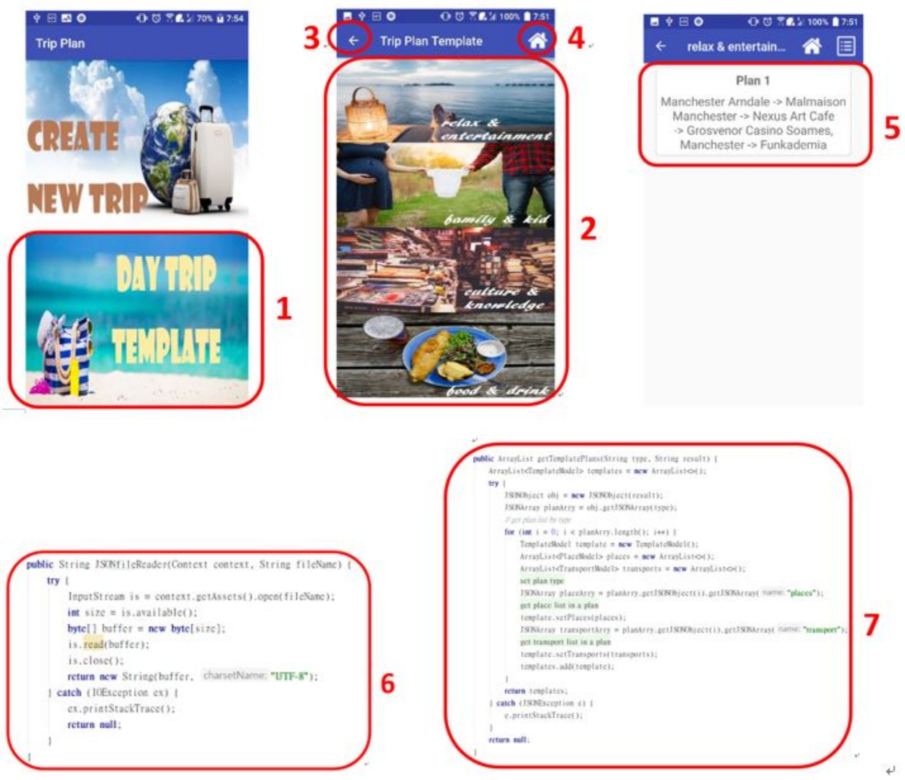
12 May, 2018 2 commits

add SimpleNLG library
Jung Huang committed 3 months ago — 5c156e05 — Browse Files

get google places api and show on map
Jung Huang committed 3 months ago — ad94be50 — Browse Files

Figure 4.4: commit history on Gitlab

4.3.1 Implementation result



Plan 1
Manchester Arndale -> Malmaison Manchester -> Nexus Art Cafe -> Grosvenor Casino Soames, Manchester -> Funkademia

```
public String JSONfileReader(Context context, String fileName) {
    try {
        InputStream is = context.getAssets().open(fileName);
        int size = is.available();
        byte[] buffer = new byte[size];
        is.read(buffer);
        is.close();
        return new String(buffer, charsetName: "UTF-8");
    } catch (IOException ex) {
        ex.printStackTrace();
        return null;
    }
}
```

```
public ArrayList getTemplatePlans(String type, String result) {
    ArrayList<TemplateModel> templates = new ArrayList<>();
    try {
        JSONObject obj = new JSONObject(result);
        JSONArray planArry = obj.getJSONArray(type);
        // get plan list by type
        for (int i = 0; i < planArry.length(); i++) {
            TemplateModel template = new TemplateModel();
            ArrayList<PlaceModel> places = new ArrayList<>();
            ArrayList<TransportModel> transports = new ArrayList<>();
            set plan type
            JSONArray placeArry = planArry.getJSONObject(i).getJSONArray( name: "places");
            get place list in a plan
            template.setPlaces(places);
            JSONArray transportArry = planArry.getJSONObject(i).getJSONArray( name: "transport");
            get transport list in a plan
            template.setTransports(transports);
            templates.add(template);
        }
        return templates;
    } catch (JSONException e) {
        e.printStackTrace();
    }
    return null;
}
```

Figure 4.5: Implementation Result for Sprint 1

1. This is home page that includes two options: create new trip and day trip template. The purpose of this Sprint is to implement day trip template.
2. The trip plan is divided into four categories: relax and entertainment, family and kid, culture and knowledge, food and drink, users can choose one of them to get a list of plans.
3. This arrow appears on every page except home page, which allows to go back to the previous page.
4. The house icon appears in every page except home page, which allows to go to home page directly.
5. This page shows after choosing relax and entertainment category, which lists several trip plans related to that category. Here only illustrates one plan as an example.

6. The trip plans are retrieved from a JSON file while this function aims to read data from that file.

7. This is the pseudocode about parsing JSON, which searches through the structure by using JSONArray and JSONObject.

4.3.2 Burn-up chart

There are five tasks in Sprint 1 that lasts fourteen days. These tasks are relatively basic and easy to complete so that all the tasks are finished on time. Figure 4.6 shows the burn-up chart for Sprint 1.

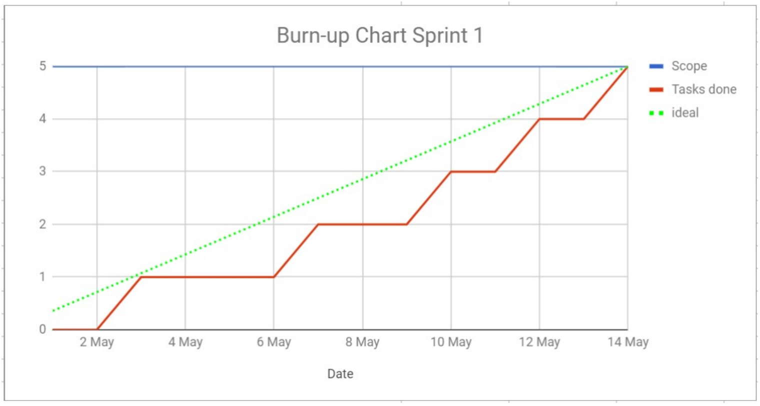


Figure 4.6: burn-up chart for Sprint 1

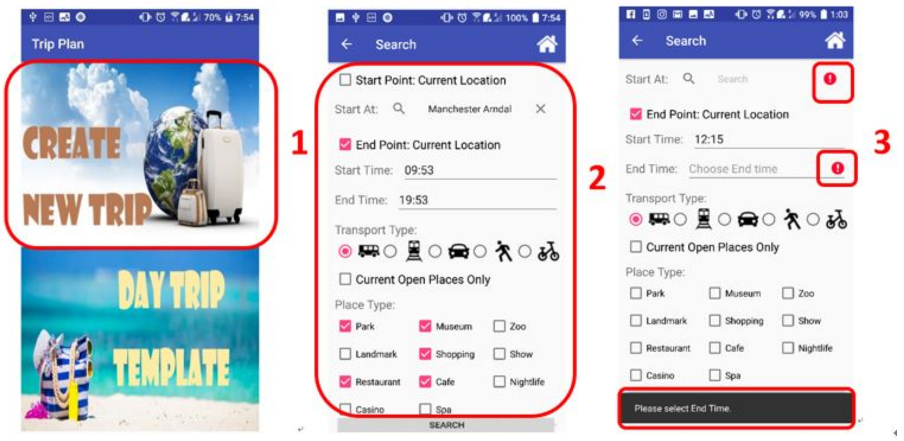
4.4 Sprint 2

In the second Sprint, the aim is to implement generating a trip plan based on user's preference that is one of the most important features for this project. The tasks include creating a page for user to filter preferences, then designing an algorithm to produce a suitable trip plan whilst the trip route is optimized, and the input data should be validated. Table 4.2 shows the Sprint backlog for Sprint 2 and the implementation results are explained as follow.

Type	User Story	Complexity	Priority
Functional	User must be able to set up multiple preferences (e.g. location, time, transport type, place type, etc.)	Medium	High
	User must be able to get trip plan based on preferences	High	High
	The trip plan route must be optimized	High	High
Non-functional	The application should validate data entered by user to prevent errors	Medium	High
	The application should generate a reasonable trip plan	High	Medium

Table 4.2: Sprint 2 backlog

4.4.1 Implementation result



Search

☐ Start Point: Current Location
Start At: Manchester Arndal
☑ End Point: Current Location
Start Time: 09:53
End Time: 19:53
Transport Type:
Current Open Places Only
Place Type: ☑ Park ☑ Museum ☐ Zoo ☐ Landmark ☑ Shopping ☐ Show ☑ Restaurant ☑ Cafe ☐ Nightlife ☐ Casino ☐ Spa

Search

Start At: Search
☑ End Point: Current Location
Start Time: 12:15
End Time: Choose End time
Transport Type:
☐ Current Open Places Only
Place Type: ☐ Park ☐ Museum ☐ Zoo ☐ Landmark ☐ Shopping ☐ Show ☐ Restaurant ☐ Cafe ☐ Nightlife ☐ Casino ☐ Spa
Please select End Time.

Figure 4.7: Implementation Result for Sprint 2

1. This is the way to enter preference page in the home page.
2. This page is preference page that includes start and end location which can be specified a point or using current location; start and end time; transport type which can be only chosen one option; whether only show the opening attractions; place type which can be selected multiple options.

3. This screenshot shows the validation for user input that exclamation mark would be shown behind errors and error message would be put in the bottom of the screen. The validation check includes that all the locations should be in Greater Manchester and cannot be empty, end time should be greater than start time and both of them cannot be empty, the time duration should be greater than one and half hour.

After pressing SEARCH button, the application starts to build trip plan that the first step is to get a list of places as ”PlaceList” based on user’s preference, then randomly choose a set of places as ”TargetPlaces”, which may be the places for the trip plan. Following that, optimize route for TargetPlaces by Genetic Algorithm and build the trip plan by Algorithm 1. Figure 4.8 shows the flow for trip plan generation and Algorithm 1 shows the pseudocode for building the trip plan. Furthermore, route optimization is implemented by using Genetic Algorithm to solve Traveling Salesman Problem depends on traveling time that Algorithm 2 shows the pseudocode for it and the flow for GA is depicted in Figure 2.3.

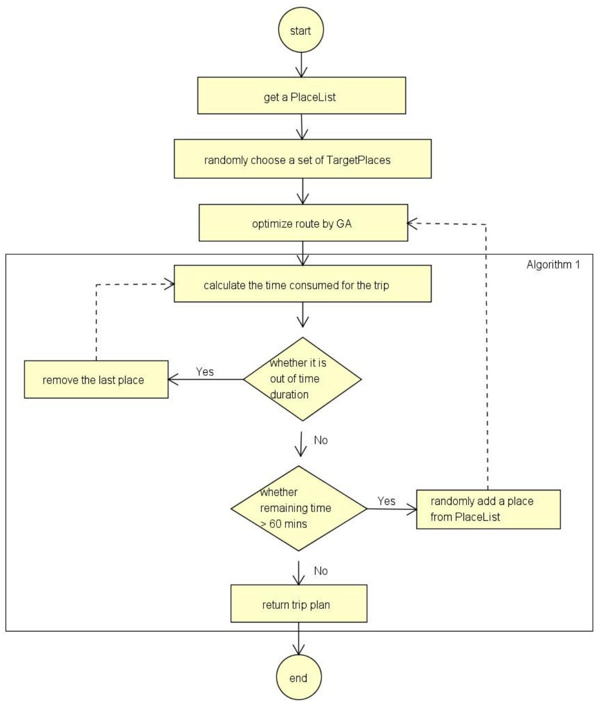

start
get a PlaceList
randomly choose a set of TargetPlaces
optimize route by GA
Algorithm 1
calculate the time consumed for the trip
whether it is out of time duration
Yes
remove the last place
No
whether remaining time > 60 mins
Yes
randomly add a place from PlaceList
No
return trip plan
end

Figure 4.8: trip plan generation flow

Algorithm 1 pseudocode for building the trip plan

```
Total Traveling Time $tt$ = traveling time between start point to first place
for all place $p$ in TargetPlaces set $TP$ do
    $t$ = traveling time between $p(i)$ and $p(i+1)$
    $tt+ = (t+60)$                                ▷ 60 is the staying time for each place
    if $td - tt < 0$ then                                            ▷ $td$:Trip Duration
        $tt- = (t+60)$
    else
        add place to trip plan

while the size of place set in trip plan $> 0$ do
    $tt+ =$ traveling time between last place to end point
    if $td - tt < 0$ then
        remove last place from trip plan and re-calculate time
    else if $td - tt > 60$ then
        select a new place from PlaceList and add to TargetPlaces
        optimize route for TargetPlaces
        do Algorithm 1
return trip plan
```

Algorithm 2 pseudocode for GA to solve TSP

```
Generate the initial population $P$
Compute fitness
while termination condition not met do
    select two best tours $T1, T2$ from $P$ and save to $P1$
    breed new tours through $crossover(T1, T2)$ and save them to $P2$
    mutate new tours and update $P2$
    update $P = P1 + P2$
return the best tour $T$ in $P$
```

GA is implemented by five classes:

1. PlaceModel: models a place that contains information about a place, such as name, location, type, etc., including getter and setter.

2. Tour: stores a candidate tour that is a list of places. There is also a function to calculate the total traveling time of the tour.

3. TourManager: holds the places of a tour that is a list of places. The functions include adding a place, getting a place, and clearing the list of places.

4. Population: manages a population of candidate tours that the size of the population in this project is set as fifty.

5. Genetic Algorithm: manages algorithms for evolving population that the population evolves for one hundred generations in this project.

4.4.2 Burn-up chart

This project is built through incremental approaches; therefore, there are other five tasks added, which means there are total ten tasks planned to be completed in this Sprint. These tasks are relatively complex, algorithms are spent much time to design and implement so that some of the tasks cannot be finished on time. The unfinished work will remain to the next Sprint. Figure 4.9 shows the burn-up chart for Sprint 2.

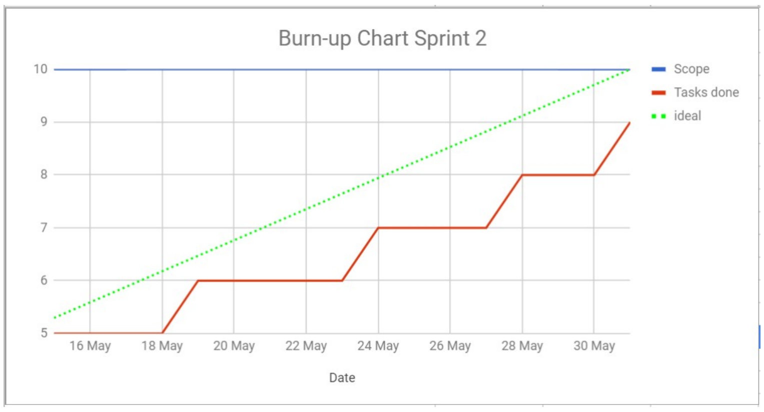


Figure 4.9: burn-up chart for Sprint 2

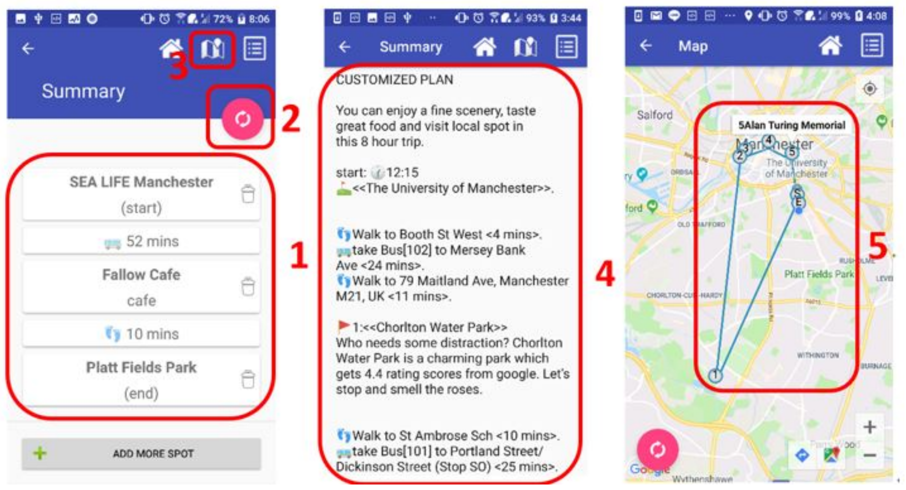
4.5 Sprint 3

In the third Sprint, the aim is to present the trip plan result in three different ways: a simple list, a detailed summary, and a map. Few information is contained in the simple list so that user can scan the whole plan quickly while the detailed summary contains far more information so that user can understand the plan comprehensively. The summary is auto-produced through SimpleNLG library that some of the sentences are constructed from a JSON file. Besides, the map depicts the location of each attraction and the route of the plan so that user can view the relative position and distance between them. Moreover, user allows to re-generate a trip plan based on the same conditions. Table 4.3 shows the Sprint backlog for Sprint 3 and the implementation results are explained as follow while the JSON file of NLG sentence will be illustrated in Appendix B.

Type	User Story	Complexity	Priority
Functional	User must be able to view trip plan in a simple list	Low	Medium
	User must be able to view trip plan in detailed summary	High	High
	User must be able to view trip plan route on map	High	High
	User must be able to re-generate a new trip plan	Low	High
Non-functional	The context in the application should be fluency, accurate, easy to understand	Medium	Medium
	Trip route should be clearly shown on the map	Medium	Medium

Table 4.3: Sprint 3 backlog

4.5.1 Implementation result



CUSTOMIZED PLAN

You can enjoy a fine scenery, taste great food and visit local spot in this 8 hour trip.

start: 12:15
<<The University of Manchester>>.

Walk to Booth St West <4 mins>.
take Bus[102] to Mersey Bank Ave <24 mins>.
Walk to 79 Maitland Ave, Manchester M21, UK <11 mins>.

1:<<Chorlton Water Park>>
Who needs some distraction? Chorlton Water Park is a charming park which gets 4.4 rating scores from google. Let's stop and smell the roses.

Walk to St Ambrose Sch <10 mins>.
take Bus[101] to Portland Street/ Dickinson Street (Stop SO) <25 mins>.

Figure 4.10: Implementation Result for Sprint 3

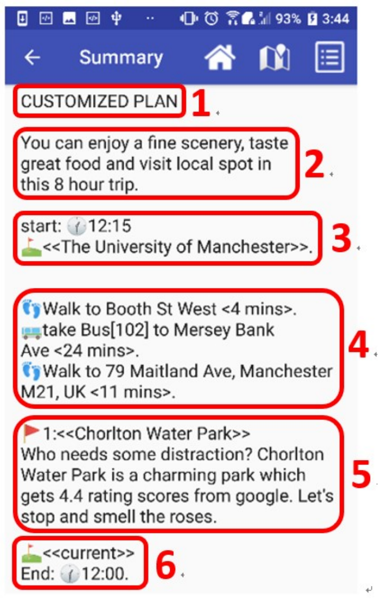
1. This is the simple list of the trip plan that only shows the name of the spots, the transport type and its duration.

2. This button is used to re-generate a new trip plan that is based on the previous preferences.

3. This button directs to the map page.

4. This is the section for detailed summary that begins with summarising the things can do during the trip, then describe the start time and location of the trip. Following that, the exhaustive steps of transport and the brief introduction of attraction are presented. Figure 4.11 and Algorithm 3 show the template and pseudocode for implementing the summary generation.

5. This is the map page that presents the trip route on a Google map by using Google Maps API. Several services of the API have been used, which include setting the location and the zoom of the map, adding icons (e.g. S,1,2,,E) to represent the sequence of the visiting places, adding polylines to connect attractions, and adding click events to the icons to show the name of that point.


Summary

CUSTOMIZED PLAN 1

You can enjoy a fine scenery, taste great food and visit local spot in this 8 hour trip. 2

start: 12:15
<<The University of Manchester>>. 3

Walk to Booth St West <4 mins>.
take Bus[102] to Mersey Bank Ave <24 mins>.
Walk to 79 Maitland Ave, Manchester M21, UK <11 mins>. 4

1:<<Chorlton Water Park>>
Who needs some distraction? Chorlton Water Park is a charming park which gets 4.4 rating scores from google. Let's stop and smell the roses. 5

<<current>>
End: 12:00. 6

Figure 4.11: template for summary generation

According to the above template, the pseudocode is described as follow:

Algorithm 3 pseudocode for summary generation

```
1. Create title of the summary
2. Things can do during the trip: You can V+N,...and V+N in this number hour trip.
3. Start time and location of the trip
repeat
    4. Steps of transport
    if there are multiple steps then
        instruction <number mins>
        take vehicle type [vehicle name] to arrival stop <number mins>
    else
        (driving/walking/bicycling) number mins
    5. Description of place:
        1st sentence: $Y/N$ question or $WHO$ question
        2nd sentence: introduce type of the place and rating score from google
        3rd sentence: activity can do in the place
until all the information is included
6. End time and location of the trip
```

4.5.2 Burn-up chart

In this Sprint, the uncompleted tasks from the previous Sprint and other six tasks should be implemented. However, there are still some works cannot be completed on time, which will keep to the next Sprint. Figure 4.12 shows the burn-up chart for Sprint 3.

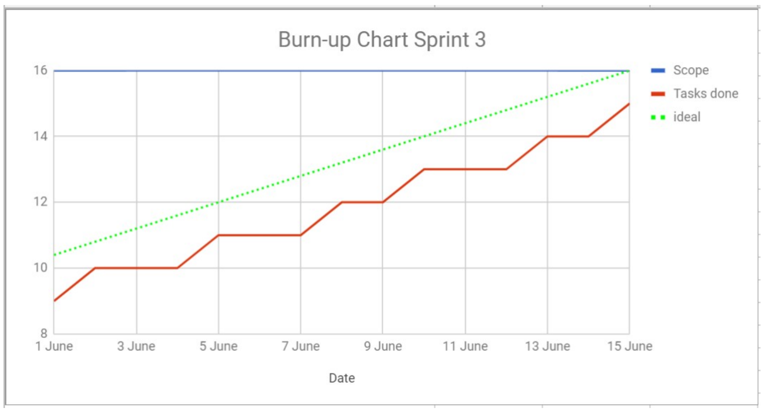


Figure 4.12: burn-up chart for Sprint 3

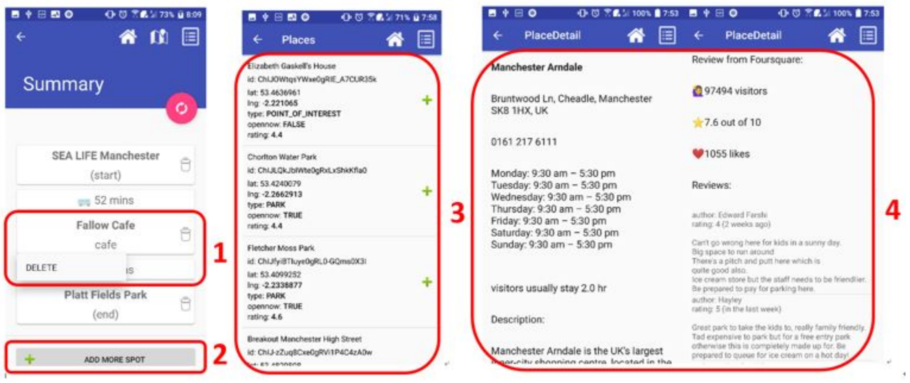
4.6 Sprint 4

In the fourth Sprint, the aim is to provide the comprehensive information for each place by collecting data from different sources and allow user to adjust the trip plan by adding and deleting places. The sources include Foursquare API and Sygic Travel API that query the result through the name of each place, and the result should be parsed before presenting. Moreover, whenever changing the trip plan, the route will be re-optimized so that the route could be determined to be the best. Table 4.4 shows the Sprint backlog for Sprint 4 and the implementation results are explained as follow.

Type	User Story	Complexity	Priority
Functional	User must be able to view the detailed information for each place (e.g. address, opening hour, rating, description, etc.)	Medium	High
	User must be able to view place detail from different sources	Medium	Medium
	User must be able to modify trip plan by deleting and adding places	Medium	Medium
	User must be able to view a list of places not in trip plan	Low	Medium
Non-functional	The application should provide enough information for traveling in Greater Manchester	Medium	Medium

Table 4.4: Sprint 4 backlog

4.6.1 Implementation result



Elizabeth Gaskell's House
id: ChIJOWtqsYWxe0gRIE_A7CUR35k
lat: 53.4636961
lng: -2.221065
type: POINT_OF_INTEREST
opennow: FALSE
rating: 4.4

Chorlton Water Park
id: ChIJLQkJbIWte0gRxLxShkKfla0
lat: 53.4240079
lng: -2.2662913
type: PARK
opennow: TRUE
rating: 4.4

Fletcher Moss Park
id: ChIJfyiBTluye0gRL0-GQms0X3I
lat: 53.4099252
lng: -2.2338877
type: PARK
opennow: TRUE
rating: 4.6

Breakout Manchester High Street
id: ChIJ-zZuqBCxe0gRVI1P4C4zA0w

Manchester Arndale

Bruntwood Ln, Cheadle, Manchester
SK8 1HX, UK

0161 217 6111

Monday: 9:30 am – 5:30 pm
Tuesday: 9:30 am – 5:30 pm
Wednesday: 9:30 am – 5:30 pm
Thursday: 9:30 am – 5:30 pm
Friday: 9:30 am – 5:30 pm
Saturday: 9:30 am – 5:30 pm
Sunday: 9:30 am – 5:30 pm

visitors usually stay 2.0 hr

Description:

Manchester Arndale is the UK's largest

Review from Foursquare:

97494 visitors

7.6 out of 10

1055 likes

Reviews:

author: Edward Farshi
rating: 4 (2 weeks ago)

Can't go wrong here for kids in a sunny day.
Big space to run around
There's a pitch and putt here which is
quite good also.
Ice cream store but the staff needs to be friendlier.
Be prepared to pay for parking here.

author: Hayley
rating: 5 (in the last week)

Great park to take the kids to, really family friendly.
Tad expensive to park but for a free entry park
otherwise this is completely made up for. Be
prepared to queue for ice cream on a hot day!

Figure 4.13: Implementation Result for Sprint 4

1. Trip plan modification is operated on the simple list. Pressing the trash can and then choosing DELETE can remove the place. Otherwise, click the name of place can direct to the PlaceDetail page.

2. Press ADD MORE SPOT button would direct to the page showing a list of places so that user can add a new spot into the trip plan.

3. This page contains several attractions, pressing + icon can add the place into the trip plan while clicking the row can direct to the PlaceDetail page.

4. This is the page to show the place details that include address, phone number, opening hour, estimated staying time retrieved from Sygic Travel API, and description, check-ins, rating, likes, feedbacks obtained from Foursquare API. For using RESTful API, the first step is to create a valid request URL that usually contains protocol, query parameters, and API key, then receiving the response as InputStream by making HTTP GET request to the given URL. Following that, the InputStream is converted into String so that it can be parsed through JSONArray and JSONObject, which is similar to paring plan template JSON file illustrated in Section 4.3.1. Figure 4.14 shows the flow for using RESTful API while Figure 4.15 shows the methods for making GET request and converting response to String.

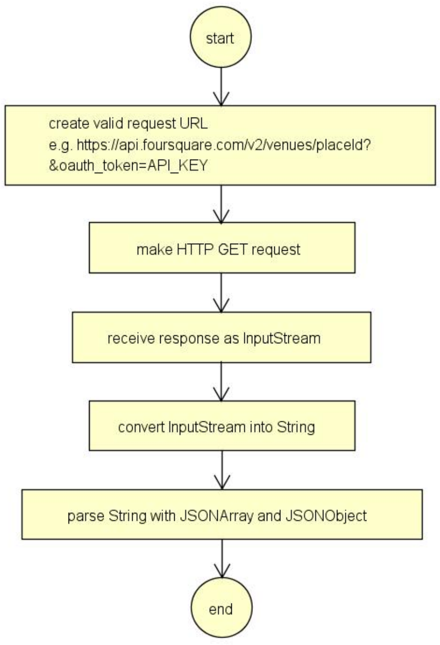
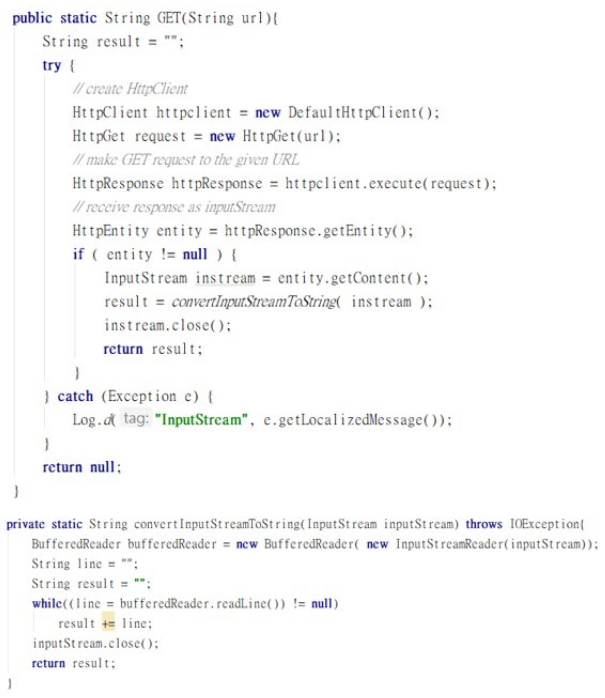

start
create valid request URL
e.g. https://api.foursquare.com/v2/venues/placeId?
&oauth_token=API_KEY
make HTTP GET request
receive response as InputStream
convert InputStream into String
parse String with JSONArray and JSONObject
end

Figure 4.14: flow for using RESTful API



```
public static String GET(String url){
    String result = "";
    try {
        // create HttpClient
        HttpClient httpclient = new DefaultHttpClient();
        HttpGet request = new HttpGet(url);
        // make GET request to the given URL
        HttpResponse httpResponse = httpclient.execute(request);
        // receive response as inputStream
        HttpEntity entity = httpResponse.getEntity();
        if ( entity != null ) {
            InputStream instream = entity.getContent();
            result = convertInputStreamToString( instream );
            instream.close();
            return result;
        }
    } catch (Exception e) {
        Log.d( tag: "InputStream", e.getLocalizedMessage());
    }
    return null;
}

private static String convertInputStreamToString(InputStream inputStream) throws IOException{
    BufferedReader bufferedReader = new BufferedReader( new InputStreamReader(inputStream));
    String line = "";
    String result = "";
    while((line = bufferedReader.readLine()) != null)
        result += line;
    inputStream.close();
    return result;
}
```

Figure 4.15: methods for making GET request and converting to String

4.6.2 Burn-up chart

There are total twenty-one tasks should be completed in this Sprint, which includes all the functional requirements. There is no unfinished task during this Sprint. Figure 4.16 shows the burn-up chart for Sprint 4.

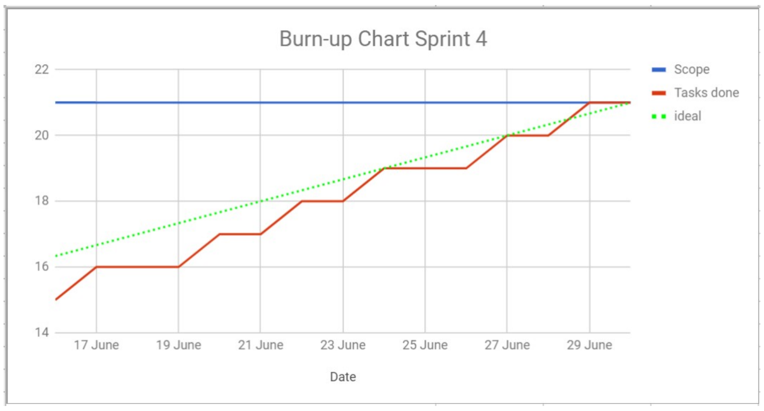


Figure 4.16: burn-up chart for Sprint 4

4.7 Sprint 5

In the final Sprint, the aim is to refine the application such as improving the performance and enhancing the user experience. The main tasks in this Sprint are to modify or refactor the existing code rather than adding new functionalities; therefore, there is no implementation result explained in this section. Table 4.5 shows the Sprint backlog for Sprint 5.

Type	User Story	Complexity	Priority
Non-functional	Design interface should be simple, clear, easy to use	Medium	Medium
	Design interface should be visual aids to facilitate its use (e.g. icons, colours, shapes, etc.)	Medium	Medium
	The terms used in design interface components (e.g. labels, buttons, tables, etc.) should be consistent among all the application and match the users vocabulary	Medium	Medium
	Design interface should display trip plan results in a complete but easy to understand manner	Medium	Medium
	Design interface should offer appropriate visualization modes for different functionalities (e.g. lists, tables, maps, etc.)	Medium	Medium
	The application should be easy and intuitive to use	Medium	Medium
	The application should be simple and appropriate for travelers who travel in Greater Manchester	Medium	Medium
	The application should notify user when errors occur using design interface	Medium	High

Table 4.5: Sprint 5 backlog

4.7.1 Burn-up chart

All of the user stories in this Sprint are non-functional, the aim is to improve the performance and user experience; therefore, the finished date is difficult to be defined. According to this, the accomplish date is estimated as every two days. Figure 4.17 shows the burn-up chart for Sprint 5.

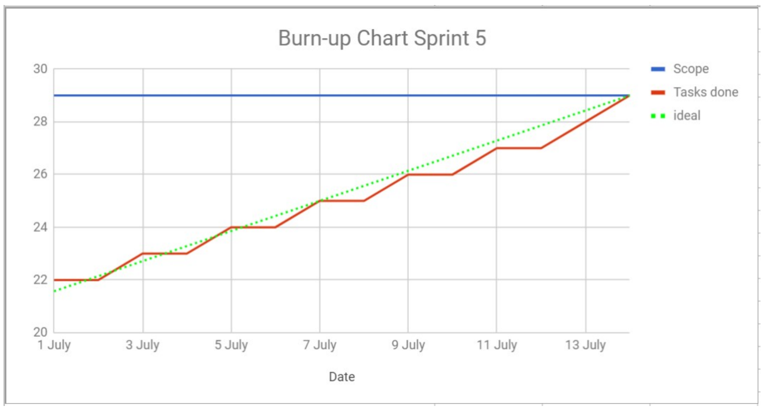


Figure 4.17: burn-up chart for Sprint 5

Chapter 5

Testing and Evaluation

This chapter presents the strategy and result of the testing and evaluation, Section 5.1 begins by introducing the test-driven development and its benefits, then describing the types of testing in Android with the pseudocode while the results of testing are explained in the subsection. This project is evaluated through a questionnaire that the evaluation methodology is described in Section 5.2 and the results are shown in the subsection.

5.1 Testing

Test-driven development is the testing strategy used in this project, which develops the system iteratively. The process starts with writing a new test that must fail because the feature is not implemented yet. After that, the feature is implemented to pass the test that the code should be exactly necessary to satisfy the specification of the test and without speculative coding. After passing all the tests, the code would be refactored and then repeating the cycle. The Android official guide [58] provides a workflow for testing application, which consists of two nested, iterative cycles. The outer cycle is a long and slow cycle that is responsible for integration tests while the inner cycle is a short and fast cycle that aims to deal with unit tests. The workflow is depicted in Figure 5.1.

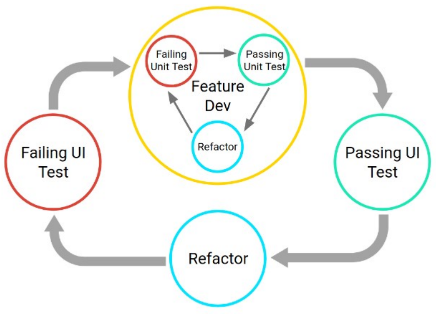


Figure 5.1: TDD workflow for Android application derived from [58]

Applying test-driven development for the project can not only save a lot of time in the long run but also release a flawless and robust application. The specific advantages include providing the rapid feedback on failures, detecting failure as early as possible, enabling safer code refactoring that getting rid of regression bugs, and minimizing the codebase since only the required code is written to make the test pass.

There are two types of testing in Android, one is local unit test and another is instrumented test. Junit is the framework used to do the unit tests that focus on testing small and specific scenario of a functionality on local development machine without accessing to any Android framework APIs. Espresso is the framework used to do the instrumented tests that are known as integration tests. These tests should be run on an Android device or an emulator by using the Android Instrumentation API via InstrumentationRegistry to control Android components, which allow to automate clicking button, typing text, swiping views, and other actions that user can perform. [58] Figure 5.2 and Figure 5.3 show the pseudocode for unit test and instrumented test.

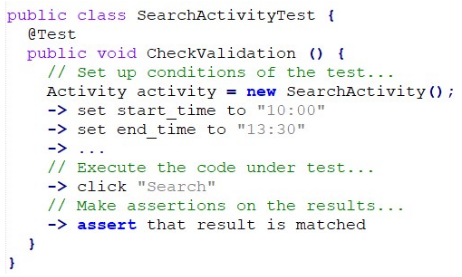

```
public class SearchActivityTest {
  @Test
  public void CheckValidation () {
    // Set up conditions of the test...
    Activity activity = new SearchActivity();
    -> set start_time to "10:00"
    -> set end_time to "13:30"
    -> ...
    // Execute the code under test...
    -> click "Search"
    // Make assertions on the results...
    -> assert that result is matched
  }
}
```

Figure 5.2: pseudocode for unit test

```
public void searchPlanByPreference() {
    // Click on the customized plan button
    onView(withId(R.id.customized_plan)).perform(click());

    // input preferences
    onView(withId(R.id.start_point)).perform(typeText("Manchester Arndale"));
    onView(withId(R.id.start_time)).perform(typeText("10:00"));
    // other preferences...

    // Search the plan
    onView(withId(R.id.search)).perform(click());

    // Verify data is displayed on screen
    onView(withItemText("Manchester Arndale")).check(matches(isDisplayed()));
}
```

Figure 5.3: pseudocode for instrumental test

5.1.1 Testing result

According to the above methodology, the unit test tests all the methods in Controller and Model module while instrumental test tests operation flows of entire application. Based on TDD, code should be implemented to pass the test and all the tests should pass at the end of coding; therefore, it is assured to release a working and less error software. As a result, all the unit tests and instrumental tests pass in this project, which means that the application works well and the implementation meets the requirements.

5.2 Evaluation

The evaluation for this project is divided into four sections: system usability, user acceptance, NLG quality, and user feedback. The first section is evaluated through System Usability Scale (SUS) that is developed by John Brooke. [59] It is a quick, dirty, and valid tool for measuring the usability, which can be used on small sample size with reliable results. There are ten questions to be asked to the user with five response options that range from strongly agree to strongly disagree while the questions are shown in Appendix C, Section 1. The second part aims to evaluate the user acceptance for trip plan that includes five questions with the same response options as section 1, which is shown in Appendix C, Section 2. The third section is about evaluating the quality of paragraph that is generated by NLG. There are eight questions with the same response options as previous sections to be asked to evaluate the fluency, readability, usefulness, etc. of the context, which are listed in Appendix C, Section 3. The goal of the final section is to obtain the feedback and comment from users so that the application can be improved, which consists of five short answer questions that are presented in Appendix C, Section 4.

5.2.1 Evaluation result

There are twenty respondents involved in this evaluation and the evaluation result is shown in four aspects. For system usability, according to the SUS, users should response each question as a scale from one to five, where one is strongly disagree and five is strongly agree. To score the result, the user response for odd questions would minus one and the user response for even questions would be subtracted from five; as a result, the score of each question would be from zero to four. All scores are added and then multiplied by 2.5 to obtain the overall result that the range is from zero to one hundred. Based on research, the average of SUS scores is sixty-eight [59] while the SUS score for this application is around seventy, which shows that the system usability is acceptable for users. For user acceptance, the aim is to evaluate that does this application helps travelers plan a trip to Greater Manchester. The result shows that most of the respondents feel satisfied with it. For NLG quality, the purpose is to evaluate the user satisfaction with the context generated by NLG. The result presents that most of respondents agree with that the context is clear and useful. Finally, the respondents also give some comments and feedbacks about this application, which are organized into future works show in Section 6.3. Furthermore, Figure 5.4 to Figure 5.6 illustrate the results of different sections.

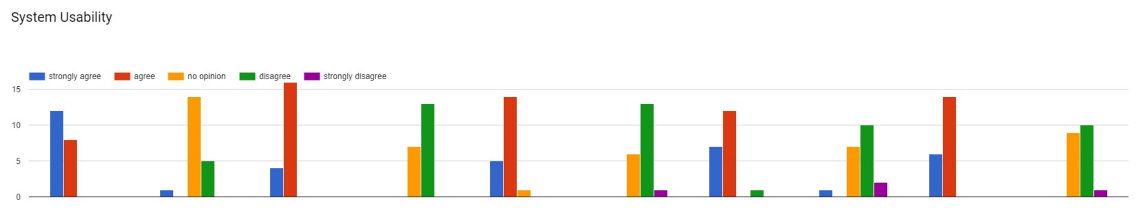


Figure 5.4: result for system usability

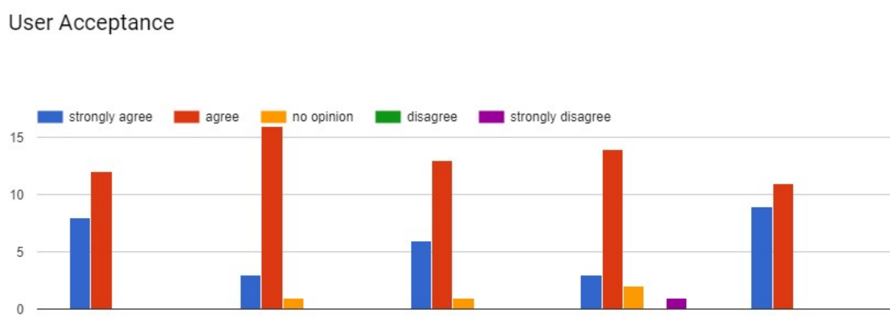


Figure 5.5: result for user acceptance

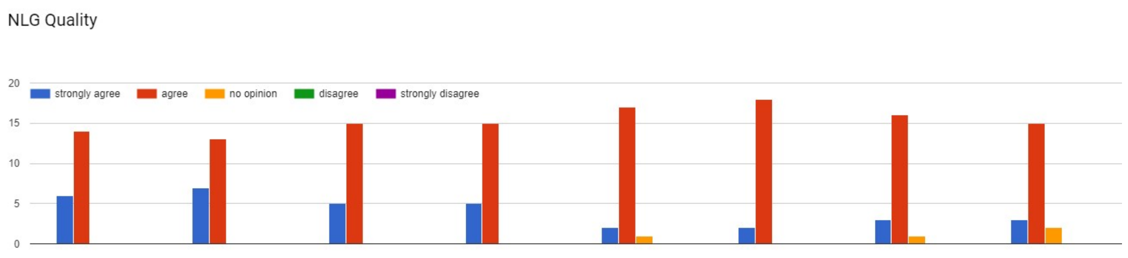


Figure 5.6: result for NLG quality

Chapter 6

Conclusions and Future Work

This chapter summarizes the project in Section 6.1 whilst mentioning the limitations for implementing the application in Section 6.2 and listing the possible future works of the project in Section 6.3.

6.1 Conclusions

The aim of the project is to create a mobile application that helps travelers make decision on their trip to Greater Manchester in a quick and simple way. In order to achieve this aim, several objectives are defined at the beginning of the project and turned into milestones as the project progressed afterward. The following describes the summary for each chapter.

- **Introduction** describe the motivation of the project and determine the related research questions. Moreover, the aim and objectives are defined to solve the research questions while project deliverable describes what kind of software to be released.

- **Background** a variety of technologies and related works are discussed in this section, including the background research on web services and web mashups that are the main data sources in this project, analysis on different types of mobile applications and platforms so that the most suitable development environment can be determined, literature reviews on NLG and algorithms to deal with TSP help to explore the best approach to implement the software, discussion on similar applications or projects helps to figure out the useful features for travelers.

- **Methodology and Application Design** before implementing the application, the development strategy, overall structure, and requirements are discussed in this section. building application through agile methodology has lots of benefits, such as responding to change and releasing working software. Furthermore, developing application with MVC pattern helps to separate the responsibilities to different modules so that it is easier to test and maintain. Moreover, functional and non-functional requirements are discussed to determine what functionalities should be done in the next stage.

- **Application Implementation** This phase introduces the implementation environment and process of developing the system. The use of Android Studio as IDE, Java as programming language, and multiple support libraries help to establish a good environment to build application. The overall activity flow and component interactions are explained before stretching into details. The whole development process is divided into five iterations, each iteration contains sprint backlog, user interface snapshots, pseudocode for algorithms, burn-up chart for progress evaluation. The first iteration focuses on establishing the project; the second and the third iteration implement the critical features of this application, which contain the logic for generating trip plan, route optimization, and producing plan summary; the fourth iteration focuses on plan modification and integrating attraction information from multiple resources; the final iteration aims to improve the usability and functionality.

- **Testing and Evaluation** TDD is the testing process used in this project, which makes sure the minimum code base and working software. Testing consists of unit tests and integration tests and all of them are passed successfully, which means that the application meets all the initially established requirements. In addition, evaluation includes system usability, user acceptance, NLG quality, and comments; the result shows that the application is acceptable and the feedbacks provide some possible improvements.

- **Conclusions and Future Work** the conclusion of each chapter is mentioned above while the limitations of the project and the possible future works that will be carried out in the following subsections.

6.2 Limitations

The main limitation for this application is that most of the web services limit the number of connections in a period of time, such as Google services restrict the connection for an ordinary user as 2500 times a day; however, this limitation can be eliminated by paying for the service. Additionally, all the data sources of this application are from web services rather than a local database; therefore, the Internet connection is very important. This is an issue in the case that users are in the area where connection is not available or very weak. Furthermore, although using web service APIs is an easy and quick way to obtain comprehensive information, there is a potential risk existing since the permanently available is not guaranteed. Finally, visiting time for each place is difficult to estimate; although Google analyses the average time from their services, it is not available for developers.

6.3 Future Works

In addition to the existing functionalities for the trip plan application, there is a variety of features that could be added to application to improve its functionality and usability. Some of the future works identified after the evaluation of this project are shown below:

- Creating a database to store user information and searching records so that the application could recommend a more suitable trip plan for user by analyzing the data from database.
- Making the trip plan more flexible; for example, let user can change the order of attractions, the transport type for different situations, the staying time for each place, and so on.
- Adding more optional searching conditions, such as budget, number of travelers, or avoiding popular time.
- Allowing user to share their trip plan as trip plan template so that users can reference the experience from others.
- Adding the route navigation for the trip plan on a map, which is an useful feature for travelers.
- Providing the timetable of public transports and some pictures for attractions.

- Finding the alternative web service APIs for the currently used one, in case the current API is disconnected.

Bibliography

[1] "Open definition." `https://opendefinition.org/od/2.1/en/`.

[2] "data.gov.uk." `https://data.gov.uk/`.

[3] "W3c web service." `https://www.w3.org/TR/ws-arch/#introduction`.

[4] H. Hamad, M. Saad, and R. Abed, "Performance evaluation of restful web services for mobile devices," vol. 1, 01 2010.

[5] R. T. Fielding, *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, 2000. AAI9980887.

[6] M. H. Bohara, M. Mishra, and S. Chaudhary, "Restful web service integration using android platform," in *2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT)*, pp. 1–6, July 2013.

[7] "Transport for greater manchester." `https://developer.tfgm.com/`.

[8] "Odata restful api." `http://www.odata.org/`.

[9] "Google places api." `https://developers.google.com/places/web-service/intro`.

[10] "Google places api example." `https://developers.google.com/places/web-service/search`.

[11] "Google maps api." `https://www.genivia.com/examples/maps/index.html`.

[12] "Sygic travel api." `https://travel.sygic.com/en`.

[13] "Foursquare api." `https://foursquare.com/`.

[14] M. M. Cruz-Cunha, J. Varajo, P. Powell, and R. Martinho, "Mobile application webservice performance analysis: Restful services with json and xml," in *ENTERprise Information Systems*, pp. 162–169, October 2011.

[15] H.-S. Zhang, Y. Zhang, Z.-H. Li, and D.-C. Hu, "Spatial-temporal traffic data analysis based on global data management using mas," *IEEE Transactions on Intelligent Transportation Systems*, vol. 5, pp. 267–275, Dec 2004.

[16] D. Fichter, "What is a mashup," in *What is a Mashup*, vol. 227, pp. 3–17, 2009.

[17] "Trendsmap." `https://www.trendsmap.com/`.

[18] "Skyscanner." `https://www.skyscanner.net/`.

[19] X. Zhiming, "Mobile app development," *Information Technology*, 2016.

[20] N. Serrano, J. Hernantes, and G. Gallardo, "Mobile web apps," *IEEE Software*, vol. 30, pp. 22–27, Sept 2013.

[21] G. Kadambari, K. Z. Han, S. G. Tilak, and V. V. Q. Huong, "Ch9: Mobile platform," in *A Fresh Graduates Guide to Software Development Tools and Technologies*, April 2011.

[22] "Android developers guide." `https://developer.android.com/guide/platform/`.

[23] "Android developers guide." `https://developer.android.com/reference/android/app/Activity`.

[24] J. Knutsen, "Web service clients on mobile android devices," *NTNU*, June 2009.

[25] N. P. Perera R, "Recent advances in natural language generation: A survey and classification of the empirical literature," *Computing and Informatics*, vol. 36, pp. 1–32, 2017.

[26] E. Dale, Robert; Reiter, "Building natural language generation systems," *Cambridge, U.K.: Cambridge University Press*, 2000.

[27] A. Gatt and E. Reiter, "Simplenlg: A realisation engine for practical applications," in *Proceedings of the 12th European Workshop on Natural Language Generation*, ENLG '09, (Stroudsburg, PA, USA), pp. 90–93, Association for Computational Linguistics, 2009.

[28] J. A. Bateman, "Enabling technology for multilingual natural language generation: The kpml development environment," *Nat. Lang. Eng.*, vol. 3, pp. 15–55, Mar. 1997.

[29] M. White, "Ccg chart realization from disjunctive inputs," in *Proceedings of the Fourth International Natural Language Generation Conference*, INLG '06, (Stroudsburg, PA, USA), pp. 12–19, Association for Computational Linguistics, 2006.

[30] R. Ehud, *Natural Language Generation*, ch. 20, pp. 574–598. Wiley-Blackwell, 2010.

[31] L. Helshani, "An Android Application for Google Map Navigation System, Solving the Travelling Salesman Problem, Optimization throught Genetic Algorithm," in *Proceedings of FIKUSZ '15*, Proceedings of FIKUSZ 2015, pp. 89–102, buda University, Keleti Faculty of Business and Management, 2015.

[32] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook, *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)*. Princeton, NJ, USA: Princeton University Press, 2007.

[33] A. H. Land and A. G. Doig, "An automatic method of solving discrete programming problems," *Econometrica*, vol. 28, no. 3, pp. 497–520, 1960.

[34] "Wiki: Branch and bound." `https://en.wikipedia.org/wiki/Branch_and_bound`.

[35] D. S. Johnson and L. A. Mcgeoch, "Table of contents," 1995.

[36] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1992.

[37] D. Whitley, "A genetic algorithm tutorial," *Statistics and Computing*, vol. 4, pp. 65–85, Jun 1994.

[38] N. Mohd Razali and J. Geraghty, "Genetic algorithm performance with different selection strategies in solving tsp," vol. 2, 01 2011.

[39] N. Silamai, N. Khamchuen, and S. Phithakkitnukoon, “Triprec: Trip plan recommendation system that enhances hotel services,” in *Proceedings of the 2017 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2017 ACM International Symposium on Wearable Computers*, UbiComp ’17, pp. 412–420, 2017.

[40] “Google trips.” `https://get.google.com/trips/`.

[41] “Manifesto for agile software development.” `http://agilemanifesto.org/`.

[42] T. Dingsyr, S. Nerur, V. Balijepally, and N. B. Moe, “A decade of agile methodologies: Towards explaining agile software development,” *Journal of Systems and Software*, vol. 85, no. 6, pp. 1213 – 1221, 2012.

[43] K. Beck, *Extreme Programming Explained: Embrace Change*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2000.

[44] R. Agarwal and D. Umphress, “Extreme programming for a single person team,” pp. 82–87, 01 2008.

[45] “Don wells.” `http://www.extremeprogramming.org/donwells.html`.

[46] “Xp process.” `http://www.extremeprogramming.org/map/project.html`.

[47] “Xp loop.” `http://www.extremeprogramming.org/map/loops.html`.

[48] H. Takeuchi and I. Nonaka, “The new new product development game,” *Harvard Business Review*, 1986.

[49] K. Schwaber, “Scrum development process,” in *Business Object Design and Implementation* (J. Sutherland, C. Casanave, J. Miller, P. Patel, and G. Hollowell, eds.), pp. 117–134, 1997.

[50] “Mvc xerox parc 1978-79.” `http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html`.

[51] G. E. Krasner and S. T. Pope, “A description of the model-view-controller user interface paradigm in the smalltalk-80 system,” vol. 1(3), 01 2000.

[52] “Systems engineering fundamentals,” 2001.

[53] K. Adams, *Non-functional Requirements in Systems Analysis and Design*, vol. 28. 05 2015.

[54] "Android developers api reference." `https://developer.android.com/reference/`.

[55] "Android developers 4.0 apis." `https://developer.android.com/about/versions/android-4.0`.

[56] J. Rumbaugh, I. Jacobson, and G. Booch, *Unified Modeling Language Reference Manual, The (2Nd Edition).* Pearson Higher Education, 2004.

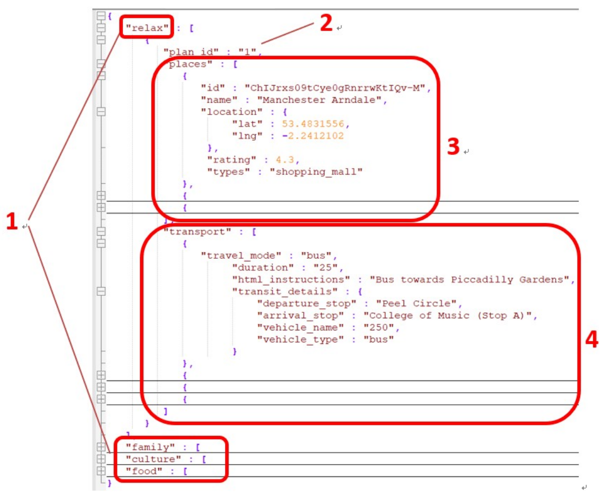
[57] J. Rumbaugh, I. Jacobson, and G. Booch, *Unified Modeling Language Reference Manual, The (2Nd Edition).* Pearson Higher Education, 2004.

[58] "Android developers guides." `https://developer.android.com/training/testing/fundamentals`.

[59] J. Brooke, *SUS: a 'quick and dirty' usability scale.* 1996.

Appendix A

PlanTemplate.json



```
{
    "relax" : [
        {
            "plan_id" : "1",
            "places" : [
                {
                    "id" : "ChIJrxs09tCye0gRnrrwKtIQv-M",
                    "name" : "Manchester Arndale",
                    "location" : {
                        "lat" : 53.4831556,
                        "lng" : -2.2412102
                    },
                    "rating" : 4.3,
                    "types" : "shopping_mall"
                },
                {
                {
            ],
            "transport" : [
                {
                    "travel_mode" : "bus",
                        "duration" : "25",
                        "html_instructions" : "Bus towards Piccadilly Gardens",
                        "transit_details" : {
                            "departure_stop" : "Peel Circle",
                            "arrival_stop" : "College of Music (Stop A)",
                            "vehicle_name" : "250",
                            "vehicle_type" : "bus"
                        }
                },
                {
                {
                {
            ]
        }
    ],
    "family" : [
    "culture" : [
    "food" : [
}
```

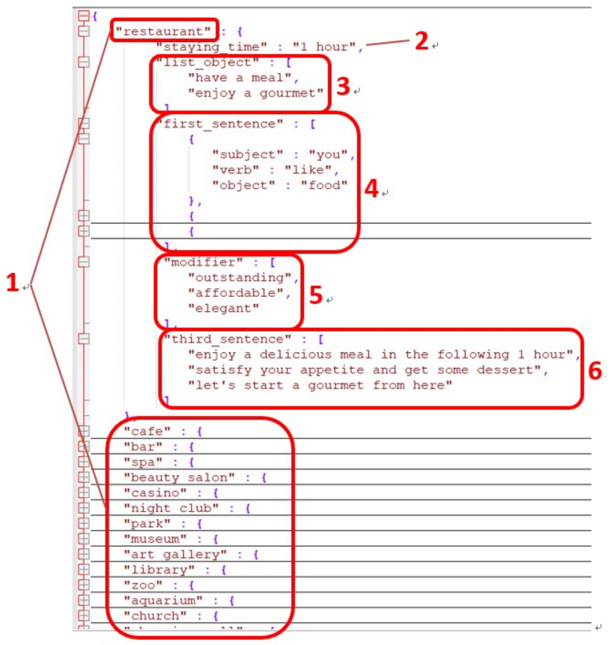
Figure A.1: JSON file for PlanTemplate

This is a json file that stores the trip templates, including:

1. Four categories as JSONArray
2. Each category contains several plans, a plan is a JSONObject
3. Each plan includes multiple places as a JSONArray and each place is a JSONObject that stores id, name, location, rating, and type.
4. Each plan also includes multiple transport steps as a JSONArray and each step is a JSONObject to present the details of instruction.

Appendix B

NLGtemplate.json



```
{
    "restaurant" : {
        "staying_time" : "1 hour",
        "list_object" : [
            "have a meal",
            "enjoy a gourmet"
        ],
        "first_sentence" : [
            {
                "subject" : "you",
                "verb" : "like",
                "object" : "food"
            },
            {
            {
        ],
        "modifier" : [
            "outstanding",
            "affordable",
            "elegant"
        ],
        "third_sentence" : [
            "enjoy a delicious meal in the following 1 hour",
            "satisfy your appetite and get some dessert",
            "let's start a gourmet from here"
        ]
    },
    "cafe" : {
    "bar" : {
    "spa" : {
    "beauty salon" : {
    "casino" : {
    "night club" : {
    "park" : {
    "museum" : {
    "art gallery" : {
    "library" : {
    "zoo" : {
    "aquarium" : {
    "church" : {
```

Figure B.1: JSON file for NLGtemplate

This is a json file that stores the elements for NLG to generate a plan summary, including:

1. Sixteen types of place, such as restaurant, museum, bar, spa, zoo, etc., different type has different description.

2. Each type is estimated its staying time.

3. Each type contains a list_object as a JSONArray that describes some activities can do in this type of place.
4. Each type has an array of JSONObjects that contain subject, verb, and object to construct question sentence for each place.
5. Each type includes an array of modifiers that would be chosen to be the adjective of a place.
6. Each type also has an array of sentences that would be selected to be the part of introduction for a place.

Appendix C

Questionnaire

There are four sections for a questionnaire that section one to section three contains several questions to be asked to the user with five response options that range from strongly agree to strongly disagree and section four consists of five optional short answer questions. The questions for each section are shown as follows.

Section1. System Usability

1. I think that I would like to use this system frequently.
2. I found the system unnecessarily complex.
3. I thought the system was easy to use.
4. I think that I would need the support of a technical person to be able to use this system.
5. I found the various functions in this system were well integrated.
6. I thought there was too much inconsistency in this system.
7. I would imagine that most people would learn to use this system very quickly.
8. I found the system very cumbersome to use.
9. I felt very confident using the system.
10. I needed to learn a lot of things before I could get going with this system.

Section2. User Acceptance

1. This app provides enough features for the trip planner.

2. The trip plan generated by this app is reasonable.
3. This app is useful for planning the trip in Greater Manchester.
4. The app does everything I would expect it to.
5. The app works the way I would want it to work.

Section3. NLG Quality

1. The prompts for input are clear.
2. The prompts for input are effective in making you want to contribute.
3. The summary of trip plan is useful.
4. The context of summary is accurate and reasonable.
5. The context of summary is easily understood.
6. The context of summary is fluency.
7. The information for trip plan is clearly presented.
8. The information for trip plan is adequate.

Section4. User Feedback

1. Did the app provide all of information you want? If not, what information do you still need?
2. What could improve the useful aspects of the app?
3. Did anything about the app confuse you? (content, layout, design, the point of the app, etc.)
4. What features should we add to improve the app?
5. Comments, opinions, suggestions

Appendix D

Gantt chart

The chart below shows the progress of the project, which includes four milestones that are represented as a star icon. Otherwise, each Sprint lasts about two weeks that the details are described in Chapter 4 while evaluation and dissertation start at the same time as soon as the application is completed. The whole project is finished at the end of August.

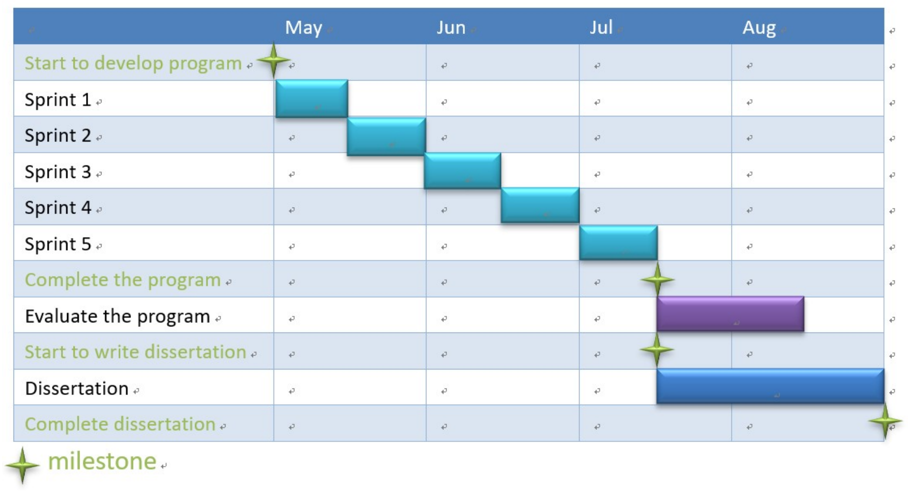


Figure D.1: Gantt chart